On the Geometry and Topology of Neural Circuits for Modular Addition

Gabriela Moisescu-Pareja[*]
McGill University, Mila

Gavin McCracken[*]
McGill University, Mila

Harley Wiltzer
McGill University, Mila

Vincent Létourneau
Université de Montréal, Mila

Colin Daniels
Independent

Jonathan Love
Leiden University

Abstract

Using tools from geometry and topology, we reveal that the circuits learned by neural networks trained on modular addition are simply different implementations of one global algorithmic strategy. We show that all architectures previously studied on this problem learn topologically equivalent algorithms . Notably, this finding concretely reveals that what appeared to be disparate circuits emerging for modular addition in the literature are actually equivalent from a topological lens. Furthermore, we introduce a new neural architecture that truly does learn a topologically distinct algorithm. We then resolve this under the lens of geometry however, and recover universality by showing that all networks learn modular addition either via 2D toroidal intermediate representations, or via combinations of certain projections of this 2D torus. Resultantly, we argue that our geometric and topological perspective on neural circuits restores the universality hypothesis.

1 Introduction

As machine learning models scale and begin to be deployed in increasing high-stakes settings, it will be imperative to develop a concrete understanding of how these models make decisions. Yet, a line of work leading up to that of Zhong et al. [1] has suggested that such a pursuit is ultimately doomed. Zhong et al. [1], in particular, focuses on the toy problem of modular addition to establish a certificate of *non-universality*: even in this simple learning problem, upon investigation of logits and embeddings, it was concluded that there are multiple distinct mechanisms by which neural networks learn to solve the task. This suggests that the identification of simple circuits in larger neural networks can be exponentially difficult; hence, their interpretability may be fundamentally unachievable.

Despite this finding, the recent work of McCracken et al. [2] found that the networks in Zhong et al. [1]'s study are all invoking an *approximate Chinese remainder theorem* to perform modular addition, suggesting that a universal interpretation may exist indeed. This work closes the apparent discrepancy by introducing a new approach for inspecting hidden representations in neural networks, focusing especially on the geometry and topology of neuron populations. Our contributions include:

A homological perspective on neural circuits. We introduce a method for prescribing a *homology* to hidden representations in neural networks. This homology precisely encodes the topology induced by the network, further encoding a rich latent metric between network inputs, and concisely informing the effective dimension of representations.

A metric in circuit space. Leveraging an understanding of the topology of the hidden representations, we construct a metric that elucidates the degree to which trained neural networks differ as

[*]Equal contribution. {gabriela.moisescu-pareja, gavin.mccracken}@mail.mcgill.ca

Mechanistic Interpretability Workshop at NeurIPS 2025.

computational circuits. Specializing to the case of modular addition, this metric takes the form of the *torus distance*. Through extensive empirical testing, we find that the "clock" and "pizza" networks analyzed by Zhong et al. [1] are indistinguishable by our metric, in correspondence with the findings of McCracken et al. [2]. Simultaneously, we show that our metric *can* indeed separate these circuits from others that are truly distinct.

Altogether, our work finds that inspection of hidden representations from a particular geometric lens successfully characterizes circuits for modular addition. Ultimately, this result suggests that the identification of particular geometric and topological structures can unlock the interpretability of increasing large models. In the pursuit of ensuring the reliability and safety of machine learning decision-makers when the stakes are high, we may not be doomed after all.

2 Related Work

Modular addition has become the standard testbed for toy interpretability settings [3–6, 2, 7–10], illuminating phenomena like *grokking* [11]. This task is both non-linearly separable and mathematically well understood, making it ideal for asking: "*What exactly do neural networks learn, and how is that computation represented internally?*"

Two influential works stand out. First, [3] reverse-engineered transformers trained on modular addition and described their internal computations. They also introduced *progress measures* to track grokking. Building on this, [4] claimed that the algorithm generalized to all group multiplications. Second, [1] modified the transformer from [3] by interpolating between fixed (MLP-like) and learnable (transformer-like) attention. Their model appeared to learn a distinct circuit, the *Pizza*, in contrast to the *Clock* described by [3], and they proposed metrics to separate the two.

Replications underscore the brittleness of such interpretations. For instance, [12] took the exact experimental setup of [4] and showed that the generalization claim of [4] didn't apply for the symmetric group, uncovering a different interpretation entirely. Returning to modular addition, we replicated and extended the setup of [1] and likewise reached different conclusions than theirs and find that their models in fact learn only one circuit.

Finally, [2] showed that across architectures (MLPs and transformers), networks converge to a divide-and-conquer algorithm resembling an *approximate Chinese Remainder Theorem* (aCRT). They found first-layer neurons are well fit by degree-1 sinusoids, with later layers requiring degree-2, in contrast to the interpretation of [3] which used degree-2 sinusoids for all layers.

While most work has focused on reverse-engineering specific algorithms in modular addition, relatively little has been done to systematically compare neural representations themselves. Tools from other domains–such as distributional hypothesis testing and topological data analysis (TDA)–offer complementary ways to characterize representations and may enrich mechanistic interpretability. For instance, distributional methods such as maximum mean discrepancy (MMD) [13] are rarely used in mechanistic interpretability, though widely applied to detect shifts, align domains [14, 15], and test fairness [16, 17]. Topological data analysis (TDA) offers a complementary view: [18] used persistent homology to track how network layers preserve or distort input topology, and [19] surveyed TDA tools such as persistent homology and Mapper for analyzing architectures, decision boundaries, representations, and training dynamics.

3 Background and setup

We consider various neural network architectures for the task of modular addition, which means predicting the map $(a, b) \mapsto a + b \mod n$ for $a, b \in \mathbb{Z}_n$. For the sake of this paper, we fix $n = 59$, following [1]. All architectures begin by embedding the inputs a, b to vectors $\mathbf{E}_a, \mathbf{E}_b \in \mathbb{R}^{128}$ using a shared (learnable) embedding matrix. The architectures differ in how the embeddings are then processed: **MLP-Add** immediately passes $\mathbf{E}_a + \mathbf{E}_b$ through an MLP, **MLP-Concat** immediately passes the concatenation $\mathbf{E}_a \oplus \mathbf{E}_b \in \mathbb{R}^{256}$ through an MLP, and **Attention 1.0** and **Attention 0.0**, introduced by [1] pass $\mathbf{E}_a, \mathbf{E}_b$ through a self-attention layer before the MLP. Particularly, **Attention 1.0** uses the standard scaled softmax attention, while **Attention 0.0** uses a fixed, constant attention matrix (in effect having disabled learnable attention). These transformer-based architectures correspond to networks associated with the *Clock* [3] and *Pizza* [1] interpretations, respectively.

3.1 Previous works' interpretations of neural networks trained on modular addition

[3] and [1] discovered two distinct, architecture-specific interpretations for the *circuits* learned by neural networks trained to solve modular addition: the *Clock* and the *Pizza*, respectively. They provide analytical forms for the learned structures and key steps of the circuits. Both reverse engineer the computation of a particular circuit *circuit* through the network, which is conditioned on a frequency f . Both circuits begin by embedding each token $a, b \in \mathbb{Z}_p$ (for p prime) on a circle:

$$\mathbf{E}_a = [\cos(2\pi fa/n), \sin(2\pi fa/n)], \quad \mathbf{E}_b = [\cos(2\pi fb/n), \sin(2\pi fb/n)]. \tag{1}$$

What distinguishes them is how the embeddings are *transformed* post-attention. Treating the attention as a blackbox and looking at its output $\mathbf{E}_{ab}$, we see the two stories. **Clock** [3] (associated with networks $\alpha = 1.0$ networks) computes the *angle sum* on the circle. The network learns representations of the form:

$$\mathbf{E}_{ab} = [\cos(2\pi f(a+b)/n), \sin(2\pi f(a+b)/n)], \tag{2}$$

encoding the modular sum as a point on the unit circle. This requires second-order interactions (e.g. multiplying embedding components through sigmoidal attention). **Pizza** [1] (associated with $\alpha = 0.0$ networks), in contrast, $\mathbf{E}_{ab}$ adds the embeddings directly as $\mathbf{E}_a + \mathbf{E}_b$, giving:

$$\mathbf{E}_{ab} = [\cos(2\pi fa/n) + \cos(2\pi fb/n), \sin(2\pi fa/n) + \sin(2\pi fb/n)], \tag{3}$$

producing a *vector mean* on the circle. This is entirely linear in the embeddings. McCracken et al. [2] proposed the network-level aCRT algorithm, describing how clusters of different frequencies interact to compute modular addition. In this algorithm, first-layer neurons obey the **simple neuron model**, where on input (a, b) a neuron of frequency f has pre-activations (corresponding to the value before applying ReLU):

$$N(a, b) = \cos(2\pi fa/n + \phi_a) + \cos(2\pi fb/n + \phi_b), \tag{4}$$

where frequencies f and phases ϕ_a, ϕ_b are learned across training. [2] empirically verify the simple neuron model across MLPs and Transformers. This model agrees with the features assumed in the theoretical works of [5, 6]. This **simple neuron model**, composed of sinusoidal units, provides a robust approximation of learned features, and explains how the features collectively implement a CRT-like computation. While this analytically explains the features, it leaves open how these features are spatially organized and how different architectures converge to this form.

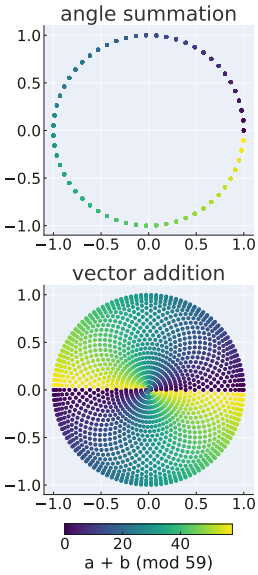

Figure 1: Clock and Pizza's analytical forms visualized, with frequency assumed to be $f = 1$ for simplicity. Each point corresponds to a pair (a, b) after being transformed by the corresponding analytical form and is colored by its sum $(a + b) \mod 59$.

Neuron remapping. For a simple neuron of frequency f, we define a canonical coordinate system via the mapping:

$$(a, b) \mapsto (a \cdot d, b \cdot d), \qquad \text{where } d := \left(\frac{f}{\gcd(f, n)}\right)^{-1} \mod \frac{n}{\gcd(f, n)}. \tag{5}$$

This inverse is the modular multiplicative inverse, i.e. for any $\mathbb{Z}_k$ let $x \in \mathbb{Z}_k$. Its inverse x^{-1} exists if $\gcd(x, k) = 1$ and gives $x \cdot x^{-1} \equiv 1 \mod k$. Here gcd is the standard procedure of finding the greatest common divisor between two integers. This normalizes inputs relative to the neuron's periodicity and allows for qualitative and quantitative comparisons.

3.2 Interpretability metrics of Zhong et al. [1]

Gradient symmetricity measures, over some subset of input-output triples (a, b, c), the average cosine similarity between the gradient of the output logit $Q_{(a,b,c)}$ with respect to the input embeddings of a and b. For a network with embedding layer $\mathbf{E}$ and a set $S \subseteq \mathbb{Z}_p^3$ of input-output triples:

$$s_g = \frac{1}{|S|} \sum_{(a,b,c) \in S} \text{sim}\left(\frac{\partial Q_{abc}}{\partial \mathbf{E}_a}, \frac{\partial Q_{abc}}{\partial \mathbf{E}_b}\right)$$

where $\mathrm{sim}(u, v) = \frac{u \cdot v}{\|u\|\|v\|}$ is the cosine similarity. It is evident that $s_g \in [-1, 1]$.

Distance irrelevance quantifies how much the model's outputs depend on the distance between a and b. For each distance d, we compute the standard deviation of correct logits over all (a, b) pairs where $a - b = d$ and average over all distances. It's normalized by the standard deviation over all data.

Formally, let $L_{i,j} = Q_{ij,i+j}$ be the correct logit matrix. The distance irrelevance q is defined as:

$$q = \frac{\frac{1}{p} \sum_{d \in \mathbb{Z}_p} \mathrm{std}(\{L_{i,i+d} | i \in \mathbb{Z}_p\})}{\mathrm{std}(\{L_{i,j} | i, j \in \mathbb{Z}_p\})}$$

where $q \in [0, 1]$, with higher values indicating greater irrelevance to input distance.

3.3 Topology

We use **Betti numbers** from algebraic topology to distinguish the structure of different stages of circuits across layers. The k-th Betti number β_k counts k-dimensional holes: β_0 counts connected components, β_1 counts loops, β_2 counts voids enclosed by surfaces. For reference, a disc has Betti numbers $(\beta_0, \beta_1, \beta_2) = (1, 0, 0)$, a circle has $(1, 1, 0)$, and a 2-torus has $(1, 2, 1)$.

We compute these using **persistent homology**, applied to point clouds constructed from intermediate representations at different stages of the circuit, as well as the final logits. This yields a compact topological signature that captures how the geometry of these representations evolves across layers, helping us identify when the underlying structure resembles a disc, torus, or circle. We use the Ripser library for these computations [20–22].

4 Hypothesis: modular addition as a factored map from the torus to the circle

Modular addition is the function $\mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ sending the pair (a, b) to $c = a + b \mod n$. Geometrically, we may embed $a \in \mathbb{Z}_n$ on the unit circle $\mathbb{R}^2$ via $\mathbf{E}_a = (\cos(2\pi a/n), \sin(2\pi a/n))$. So the product space $\mathbb{Z}_n \times \mathbb{Z}_n$ embeds into $\mathbb{R}^4$ as a discretized torus, parameterized by

$$(a, b) \mapsto (\cos u, \sin u, \cos v, \sin v), \quad u = 2\pi a/n,\ v = 2\pi b/n.$$

In this embedding, modular addition corresponds to the following map from the torus to the circle:

$$(x_1, x_2, x_3, x_4) \mapsto (x_1 x_3 - x_2 x_4, x_1 x_4 + x_2 x_3).$$

Parameterizing by angles, this becomes the familiar trigonometric identity

$$(\cos u, \sin u, \cos v, \sin v) \mapsto (\cos(u + v), \sin(u + v)).$$

We claim that networks we study are approximating this specific geometric map from a torus in $\mathbb{R}^4$ to a circle in $\mathbb{R}^2$. In Section 3.1 we saw that the "clock" and "pizza" interpretations included learned embeddings of the form of $\mathbf{E}_a$. Taken together, these embeddings define a torus $\mathbf{T}^2$ as the input representation space. **Our hypothesis** is that architectures (MLP-Add, Attention 0.0 and 1.0, MLP-Concat) do not learn fundamentally different solutions; rather they factor the same torus-to-circle map via different intermediate representations. Fig. 4 shows factorizations of this map.

4.1 Qualitative analysis of intermediate representations

In this section, we provide suggestive evidence that "clock" and "pizza" networks are geometrically almost indistinguishable: they correspond to a disc coming from "vector addition on the circle". We study 1-hidden layer versions of MLP-Add, Attention 0.0 [1, "pizza"], Attention 1.0 [1, "clock"], and MLP-Concat. In all networks, we cluster neurons together and study the entire cluster at once [2]. This is done by constructing an $n \times n$ matrix, with the value in entry (a, b) corresponding to the preactivation value on datum (a, b). A 2D Discrete Fourier Transform (DFT) of the matrix gives the key frequency f for the neuron. The cluster of preactivations of all neurons with key frequency f is the $n^2 \times |\text{cluster } f|$ matrix, made by flattening each neurons preactivation matrix and stacking the resulting vector for every neuron with the same key frequency.

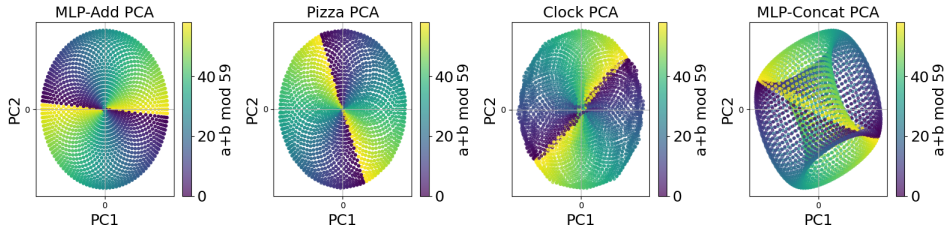

Figure 2: PCA of neuron pre-activations for a single frequency cluster f across architectures: MLP-Add ($f = 27$), Pizza ($f = 17$), Clock ($f = 21$), MLP-Concat ($f = 22$). Each point is an input (a, b), colored by $(d \cdot a + d \cdot b) \bmod 59$ corresponding to the network's output (see Section 3.1). Pizza and Clock are nearly identical to each other and to MLP-Add, but differ strongly from MLP-Concat.

Principal component analysis (PCA). We apply PCA to each cluster matrix and project the n^2 datapoints, for each input (a, b), onto the first two principal components. To compare clusters of different frequencies, we use the remapping from [2] (Section 3.1), which normalizes all clusters to frequency 1. This can be seen in Figure 2 where despite pre-activations coming from different frequency clusters, the color grading is normalized to frequency 1. The figure shows that Pizza (Attention 0.0), Clock (Attention 1.0), and MLP-Add's pre-activations each collapse to the same 2D disc, differing only by rotation; for these networks, the first two principal components explain more than 99% of the variance, implying the cluster has a 2D structure. These clusters have negligible activation near $(0, 0)$ in the PCA plane, since neither principal component contributes strongly at that point. In contrast, MLP-Concat is fundamentally different: its pre-activations form a 4D structure resembling a torus, where four principal components each explain about 25% of the variance, and no 2D projection fully captures the structure. Importantly, the 4D embedding has no datapoints at $(0, 0, 0, 0)$, indicating that the cluster activates on all points.

Distribution of post-ReLU activations. To probe how neurons activate, in Figure 3 we remap each neuron to frequency 1, using the procedure of [2] (Section 3.1), and compute the sum of post-ReLU activations across each frequency cluster. The figure shows that in MLP-Add, Pizza (Attention 0.0), and Clock (Attention 1.0), clusters activate most strongly along the diagonal where the two inputs align ($a \approx b$), with activation strength decreasing smoothly as the distance from the diagonal grows. This implies that neurons in these networks adopt phases such that they fire maximally when the two embedded inputs are in phase ($\phi_a = \phi_b$). In contrast, MLP-Concat's cluster activates more uniformly across the input space, reflecting the different 4D geometry already seen in PCA.

These results give a geometric explanation for the observation of [1]: the apparent $a - b$ dependency in Pizza logits arise because neurons are phase-aligned, so they concentrate activation when inputs coincide and the strength decreases smoothly as the difference between a and b grows. Since Clock also exhibits this behaviour, we expect that it also has an $a - b$ dependency and its activations, thus intermediate representation, resemble that of Pizza. This is surprising, because [1] identified an $a - b$ dependency as the defining feature of Pizza circuits, for which they defined their distance irrelevance metric. Our results show that this dependency arises just as strongly in Clock circuits, meaning both architectures share the same underlying geometric mechanism. **Given these observations, we hypothesize that the structure of the intermediate representations depends on the distribution of phases.** Under the simple neuron model, this aligns with where the neuron activates most strongly.

Phase Alignment Distributions. To characterize the topological similarity of the learned representations across these networks at a more fine-grained level, we propose yet another representation: the Phase Alignment Distribution (PAD). To a given architecture, a PAD is a distribution over $\mathbb{Z}_n \times \mathbb{Z}_n$. Samples of this distribution are drawn as follows:

1. Sample a random initialization (e.g., random seed) and train the network.
2. From the resulting trained network, sample a neuron uniformly.
3. Return the pair $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n$ that achieves the largest activation in the resulting neuron.

A PAD illustrates, across independent training runs and neuron clusters, how often activations are maximized on the $a = b$ diagonal—that is, it depicts how often learned phases align. Even

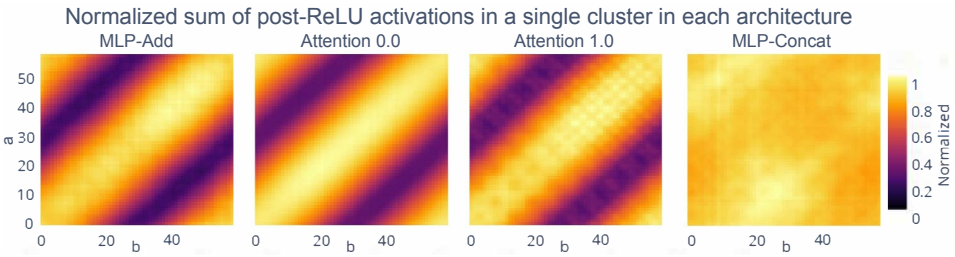

Figure 3: Given a trained network, a particular frequency cluster f is chosen and the post-ReLU activations of all neurons in that cluster are summed over all (a, b) and then the sum is normalized. This gives a summary of where a cluster activates most strongly. We visualize this summary for a single cluster in each network: MLP-Add's cluster is $f = 27$, Attention 0.0's cluster is $f = 17$, Attention 1.0's cluster is $f = 21$ and MLP-Concat's cluster is $f = 22$. Clusters in MLP-Add, Attention 0.0 and 1.0 activate strongest on (a, b) with a close to b: the activation strength decreases with distance from $a = b$; all three models have very similar behaviour. Clusters in MLP-Concat activate almost uniformly in strength (equivariantly). The Attention 0.0 and 1.0 models are model checkpoints from [1], where the Attention 0.0 is identified as a Pizza and Attention 1.0 identified as a Clock by their metrics.

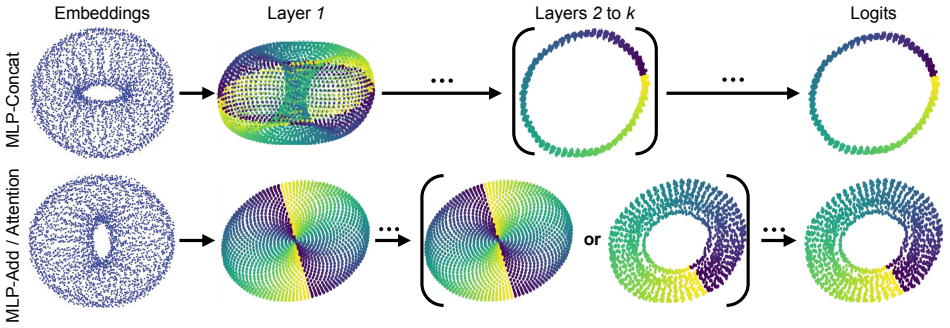

Figure 4: Different factorizations of the torus-to-circle map. We find first-layer intermediate representations to be either a torus or a disc (resembling vector addition on the circle). Later layers can construct a circle, and the logits approximate a circle.

beyond inspecting the proximity of samples to this diagonal, we propose to compare the PADs of architectures according to a metrics on the space of distributions over $\mathbb{Z}_n \times \mathbb{Z}_n$, giving an even more precise comparison. In the following section, we will provide estimates of the PADs for the aforementioned architectures, as well as PAD distances under the *maximum mean discrepancy* [13, MMD]—a family of metrics with tractable unbiased sample estimators.

As suggested from our preliminary expositions in this section, our hypothesis is that PADs will indicate that Clock and Pizza architectures learn nearly-topologically-identical circuits. Our approach for accomplishing this will be based on the realization that the MLP-Add architecture has a PAD supported entirely on the positive diagonal (this is ensured under the simple neuron model [2], since the addition of sinusoids of the same frequency will achieve the largest value when the phases align). By comparing the PADs of the Clock and Pizza to that of MLP-Add, we expect to find roughly no difference (w.r.t. a distributional metric). Moreover, we hypothesize that the PAD of the MLP-Concat architecture will be distinct, as a result of its higher-dimensional representation—this would demonstrate effectively that topological analysis can distinguish the resulting circuits.

5 Key results: attention 0.0 and 1.0 models are both *almost* vector addition

5.1 PAD shows that MLP-Add, Attention 0.0 (Pizza) and Attention 1.0 (Clock) are the same

We study 703 trained one-hidden-layer networks drawn from our four architectures: MLP-Add, Attention 0.0 (Pizza), Attention 1.0 (Clock) and MLP-Concat. Our goal is to show that the Attention

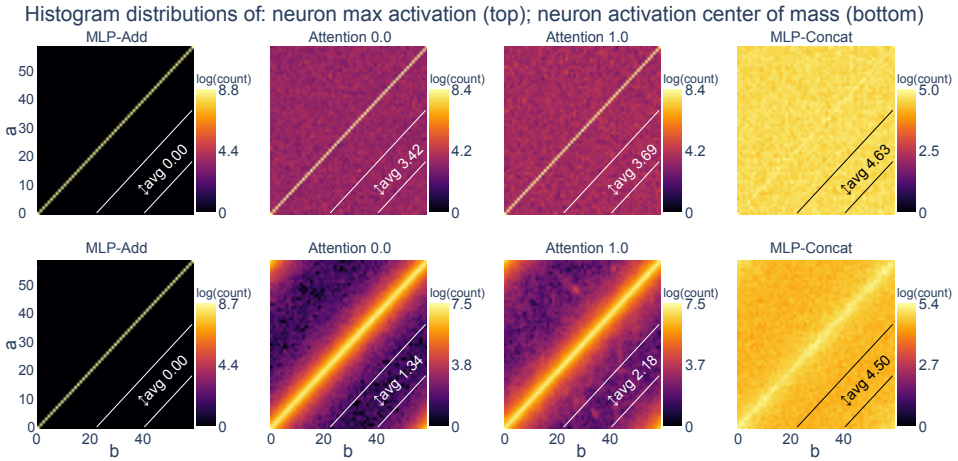

Figure 5: Log-density heatmaps for the distribution of neuron maximum activations (top) and activation center of mass (bottom) across 703 trained models. Attention 0.0 and 1.0 architectures exhibit modest off-diagonal spread compared to MLP-Add, but remain constrained by architectural bias toward diagonal alignment. The maximum mean discrepancy scores between Attention 0.0 and 1.0 are 0.0237 and 0.0181 in rows 1 and 2 respectively, indicating they are very similar distributions.

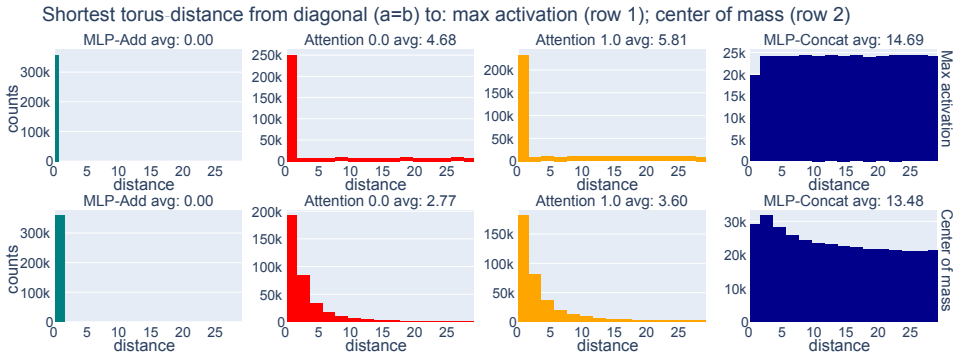

Figure 6: Histograms of torus-distance from each neuron's phase to the diagonal $a = b$, across 703 trained models. MLP-Add neurons align perfectly with the diagonal, Attention 0.0 and 1.0 show increasing off-diagonal spread, and MLP-Concat exhibits broadly distributed activations on the torus.

models are nearly equivalent to MLP-Add while being clearly distinct from MLP-Concat. Recall that MLP-Add bypasses the attention mechanism and directly sums the input embeddings, yielding a pre-ReLU representation that corresponds analytically to vector addition on the circle. Following prior work on one-layer networks [3, 1], and building on the empirical validation of the simple neuron model (Eq. 4) in [2], the only degree of freedom within a frequency cluster is the pair of learned phases (ϕ_a, ϕ_b). In practice, raw activations can be noisy, so rather than fitting sinusoids directly, we assume the simple neuron model and identify the phase either by the point of maximal activation or by the activation's center of mass.

Figure 5 shows PAD plots across architectures. We note that MLP-Add, Attention 0.0, Attention 1.0 are very concentrated on the diagonal, while MLP-Concat is not. To further quantify this, we propose the torus distance, which is the discrete graph distance from a point (a, b) on the torus to the $a = b$ line. Figure 6 quantifies this with a histogram of torus distances to the $a = b$ line. We see that Attention 0.0 and 1.0 are almost indistinguishable and both very similar to MLP-Add; moreover, our metric successfully discerns these models from MLP-Concat.

Table 1 shows the PAD distances under the MMD distance. All comparisons are statistically significant (p-values ≈ 0). We see that Attention 0.0 and 1.0 are extremely close to each other, MLP-Add lies moderately close to both, and MLP-Concat is strongly separated from all others.

Table 1: Gaussian-kernel Maximum Mean Discrepancies (MMD) [13] and permutation p-values between the empirical distributions shown in Figure 5. For each architecture comparison, we sampled 20,000 points from each empirical distribution (derived from histogram-based neuron statistics), then computed the unbiased Gaussian-kernel MMD with a bandwidth chosen via the pooled median heuristic. Significance was assessed using 50,000 permutation tests per comparison.

(a) Row 1: Max activation

Description	MMD	p-value	Interpretation
MLP-Add vs Attention 0.0	0.0968	0.0000	Moderate difference; highly significant
MLP-Add vs Attention 1.0	0.1239	0.0000	Clear difference; highly significant
MLP-Add vs MLP-Concat	0.2889	0.0000	Very strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.0338	0.0000	Subtle difference; highly significant
Attention 0.0 vs MLP-Concat	0.1987	0.0000	Strong difference; highly significant
Attention 1.0 vs MLP-Concat	0.1723	0.0000	Strong difference; highly significant

(b) Row 2: Center of mass

Description	MMD	p-value	Interpretation
MLP-Add vs Attention 0.0	0.0583	0.0000	Small difference; highly significant
MLP-Add vs Attention 1.0	0.0689	0.0000	Moderate difference; highly significant
MLP-Add vs MLP-Concat	0.2614	0.0000	Very strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.0210	0.0084	Subtle difference; highly significant
Attention 0.0 vs MLP-Concat	0.2126	0.0000	Strong difference; highly significant
Attention 1.0 vs MLP-Concat	0.1947	0.0000	Strong difference; highly significant

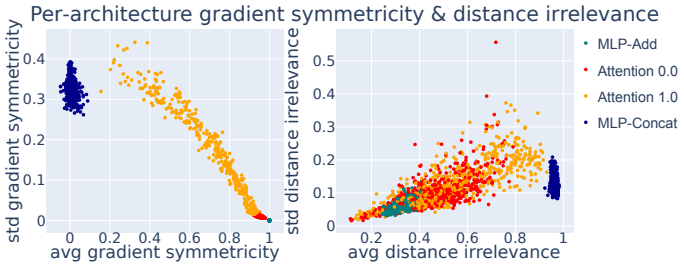

Figure 7: Evaluation of gradient symmetricity (left) and distance irrelevance (right). Each point shows the average (avg) and standard deviation (std) of one trained network. MLP-Add and MLP-Concat lie at nearly opposite extremes, while attention 0.0 and 1.0 overlap substantially. Gradient symmetricity separates Attention 1.0 better, but neither metric **always** distinguishes between Attention 1.0 and 0.0.

Figure 7 shows the mean and standard deviation of the gradient symmetricity and distance irrelevance metrics from [1] (Section 3.2). Unlike [1], who report gradient symmetricity results over a randomly selected subset of 100 input-output triples $(a, b, c) \in \mathbb{Z}_p^3$, we compute the metric exhaustively across **all** $59^3 = 205,370$ triples to add accuracy.[2].

MLP-Add and MLP-Concat cluster on opposite extremes, implying the metrics just identify whether neurons have phases $\phi_a \neq \phi_b$. MLP-Add models have high gradient symmetricity and low distance irrelevance and MLP-Concat models have low gradient symmetricity and high distance irrelevance. Attention 1.0 models span a wide range between these extremes depending on two factors: 1) how well the frequencies they learned intersect and 2) how well neurons are able to get their activation center of mass away from the $\phi_a = \phi_b$ line. Attention 0.0 is closer to MLP-Add than Attention 1.0 because it's harder for this architecture to learn $\phi_a \neq \phi_b$. Notably, failure cases exist using both: **neither metric distinguishes between Attention 1.0 and 0.0 models.**

[2]See Appendix B.3 for the GPU-optimized procedure.

Our metric has a nice topological interpretation unlike prior metrics, which we exploit further to understand intermediate representations in multi-layer networks in the next subsection.

5.2 Homology consistently explains topological transformations in deep networks

It's the case that the homology of what networks learn gives that MLP-Add, Attention 0.0 and Attention 1.0 architectures are all making topologically equivalent computations. While the MLP-Concat model appears to be different, it's in fact just more efficient, which results from the torus already having the holes necessary to accurately project the correct answer onto the logits after just one non-linearity (see Fig. 8). It's worth noting that while discs appear to be learned in the logits, in all the cases checked by hand the discs were caused by limitations of persistent homology, which struggles to find a hole of small radius.

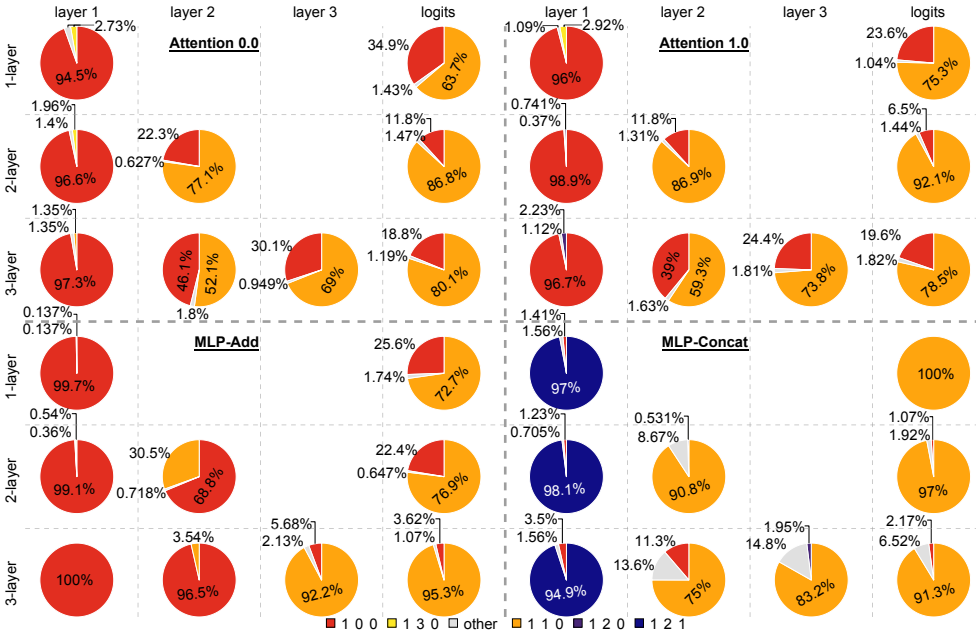

Figure 8: Betti number distributions across layers for 1-, 2-, and 3-layer models (100 seeds for each model). In layer 1, MLP-Add, Attention 0.0, and Attention 1.0 mostly yield disc-like representations, while MLP-Concat produces a torus. From the second layer onward, MLP-Add and both Attention variants converge to either a disc or a circle: the circle reflects the logits topology (correct answer), while the disc is a transient intermediate that can persist in later layers. MLP-Concat instead transitions directly to the circle. Across depth, Attention 0.0 and 1.0 are nearly identical with the latter having fewer transient discs.

6 Dsicussion and Conclusion

We've shown that architectures of various types with trainable embeddings are approximating the torus to circle map for modular addition. Thus, how circuits differ is in the way this map *factors*, i.e. the structure of the intermediate representations. As a consequence, we find that the architectures associated with the "clock" and "pizza" interpretations are factoring this map in the same way, thus they implement the same circuits. The distinction between "clock" and "pizza" is illusory, and they differ more obviously from our MLP-Concat model. We were able to detect the similarities and differences using geometric and topological methods.

Given our characterization of neural modular addition as factorizations of the torus to circle map, we discover that this can be accomplished via intermediate representations that include the torus itself or projections of it. This has interesting connections to the *manifold hypothesis* [23], which posits that data lives on a lower dimensional manifold and that neural networks will discover this underlying

low-dimensional structure. Our results give a clear demonstration of how high-level architectural choices can induce the learning of the entire manifold or a projection of it.

Finally, our analysis is limited to a single task: modular addition and a single modulus $n = 59$. While the toroidal geometry provides a useful abstraction in this domain, the feasibility of recovering the underlying manifold in more complex or real-world tasks remains to be established.

In conclusion, our work reframes the modular addition landscape: Clock and Pizza models lie on a geometric continuum shaped by architectural bias, not algorithmic difference. This opens promising directions for future research in mechanistic interpretability and theory: how do learned manifolds and their geometry govern training dynamics, generalization, and representational efficiency? Various tools from geometry and topology can be brought to bear on this problem, and many more will likely need to be developed for the nuances of neural networks, particularly the presence of *noise*.

References

[1] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S5wmbQc1We.

[2] Gavin McCracken, Gabriela Moisescu-Pareja, Vincent Letourneau, Doina Precup, and Jonathan Love. Uncovering a universal abstract algorithm for modular addition in neural networks, 2025. URL https://arxiv.org/abs/2505.18266.

[3] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

[4] Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning*, pages 6243–6267. PMLR, 2023.

[5] Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

[6] Depen Morwani, Benjamin L. Edelman, Costin-Andrei Oncescu, Rosie Zhao, and Sham M. Kakade. Feature emergence via margin maximization: case studies in algebraic tasks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=i9wDX850jR.

[7] Chun Hei Yip, Rajashree Agrawal, Lawrence Chan, and Jason Gross. Modular addition without black-boxes: Compressing explanations of mlps that compute numerical integration, 2024. URL https://arxiv.org/abs/2412.03773.

[8] Tianyu He, Darshil Doshi, Aritra Das, and Andrey Gromov. Learning to grok: Emergence of in-context learning and skill composition in modular arithmetic tasks. *arXiv preprint arXiv:2406.02550*, 2024.

[9] Tao Tao, Darshil Doshi, Dayal Singh Kalra, Tianyu He, and Maissam Barkeshli. (how) can transformers predict pseudo-random numbers? In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=asDx9sPAUN.

[10] Darshil Doshi, Aritra Das, Tianyu He, and Andrey Gromov. To grok or not to grok: Disentangling generalization and memorization on corrupted algorithmic datasets. *arXiv preprint arXiv:2310.13061*, 2023.

[11] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

[12] Dashiell Stander, Qinan Yu, Honglu Fan, and Stella Biderman. Grokking group multiplication with cosets. In *Forty-first International Conference on Machine Learning*, 2024.

[13] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[14] Muhammad Ghifary, W Bastiaan Kleijn, and Mengjie Zhang. Domain adaptive neural networks for object recognition. In *Pacific Rim international conference on artificial intelligence*, pages 898–904. Springer, 2014.

[15] Han Zhao, Remi Tachet Des Combes, Kun Zhang, and Geoffrey Gordon. On learning invariant representations for domain adaptation. In *International conference on machine learning*, pages 7523–7532. PMLR, 2019.

[16] Namrata Deka and Danica J Sutherland. Mmd-b-fair: Learning fair representations with statistical testing. In *International Conference on Artificial Intelligence and Statistics*, pages 9564–9576. PMLR, 2023.

[17] Insung Kong, Kunwoong Kim, and Yongdai Kim. Fair representation learning for continuous sensitive attributes using expectation of integral probability metrics. *IEEE transactions on pattern analysis and machine intelligence*, 2025.

[18] Archie Shahidullah. Topological data analysis of neural network layer representations, 2022. URL https://arxiv.org/abs/2208.06438.

[19] Rubén Ballester, Carles Casacuberta, and Sergio Escalera. Topological data analysis for neural network analysis: A comprehensive survey, 2024. URL https://arxiv.org/abs/2312.05840.

[20] Ulrich Bauer. Ripser: efficient computation of Vietoris-Rips persistence barcodes. *J. Appl. Comput. Topol.*, 5(3):391–423, 2021. ISSN 2367-1726. doi: 10.1007/s41468-021-00071-5. URL https://doi.org/10.1007/s41468-021-00071-5.

[21] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co)homology. *Inverse Problems*, 27(12):124003, November 2011. ISSN 1361-6420. doi: 10.1088/0266-5611/27/12/124003. URL http://dx.doi.org/10.1088/0266-5611/27/12/124003.

[22] Christopher Tralie, Nathaniel Saul, and Rann Bar-On. Ripser.py: A lean persistent homology library for python. *The Journal of Open Source Software*, 3(29):925, Sep 2018. doi: 10.21105/joss.00925. URL https://doi.org/10.21105/joss.00925.

[23] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A Additional experimental details

A.1 Training hyperparameters.

All models are trained with the Adam optimizer [24]. Number of neurons per layer in all models is 1024. Batch size is 59. Train/test split: $90\%/10\%$.

Attention 1.0

- Learning rate: 0.00075
- L2 weight decay penalty: 0.000025

Attention 0.0

- Learning rate: 0.00025
- L2 weight decay penalty: 0.000001

MLP-Add and MLP-Concat

- Learning rate: 0.0005
- L2 weight decay penalty: 0.0001

MLP-Concat

A.2 Persistent homology

For our persistent homology computations, we set the k-nearest neighbour hyperparameter to 250. Our point cloud consists of $59^2 = 3481$ points.

B Experiments and computational details

B.1 Statistical significance of the main results

B.1.1 Figure 5

We trained 703 models of each architecture, being MLP vec add, Attention 0.0 and 1.0, and MLP concat, and recorded the locations of the max activations of all neurons across all (a, b) inputs to the network. We also computed the center of mass of each neuron as this doesn't always align with the max preactivation (though it tends to be close).

B.1.2 Figure 6: Torus distance from the max activation and center of mass to the line $a = b$

We trained 703 models of each architecture with 512 neurons in its hidden layer (MLP vec add, Attention 0.0 and 1.0, and MLP concat), and recorded the a, b value of where the max activation of a neuron takes place across all (a, b) inputs to the network and all neurons. We also computed the (a, b) values for the location of the center of mass of each neuron as this doesn't always align with the max preactivation (though it tends to be close). Then we compute the shortest torus distance from the point of the max activation or the center of mass, to the line $a = b$.

B.2 Figure 7 symmetricity

MMD results for these two metrics are reported below, again showing that the distance between attention 0.0 and attention 1.0 models is small. This is the case even those these metrics were chosen to differentiate between the two architectures.

Using just the x-axis (since the y-axis on those plots is the std dev) MMD results are presented next.

We can conclude that the attention transformers are far from vector addition, and very close to each other under all metrics.

Table 2: Gaussian-kernel Maximum Mean Discrepancies (MMD) [13] and permutation p-values between the empirical distributions shown in Figure 5. For each architecture comparison, we sampled 20,000 points from each empirical distribution (derived from histogram-based neuron statistics), then computed the unbiased Gaussian-kernel MMD with a bandwidth chosen via the pooled median heuristic. Significance was assessed using 50,000 permutation tests per comparison.

(a) Row 1: Max activation

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	0.0968	0.0000	Moderate difference; highly significant
MLP vec add vs Attention 1.0	0.1239	0.0000	Clear difference; highly significant
MLP vec add vs MLP concat	0.2889	0.0000	Very strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.0338	0.0000	Subtle difference; highly significant
Attention 0.0 vs MLP concat	0.1987	0.0000	Strong difference; highly significant
Attention 1.0 vs MLP concat	0.1723	0.0000	Strong difference; highly significant

(b) Row 2: Center of mass

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	0.0583	0.0000	Small difference; highly significant
MLP vec add vs Attention 1.0	0.0689	0.0000	Moderate difference; highly significant
MLP vec add vs MLP concat	0.2614	0.0000	Very strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.0210	0.0084	Subtle difference; highly significant
Attention 0.0 vs MLP concat	0.2126	0.0000	Strong difference; highly significant
Attention 1.0 vs MLP concat	0.1947	0.0000	Strong difference; highly significant

Table 3: Gaussian-kernel Maximum Mean Discrepancies (MMD) [13] and permutation p-values between the empirical distributions shown in Figure 6. For each architecture comparison, we sampled 2000 points from each empirical distribution (derived from histogram-based neuron statistics), then computed the unbiased Gaussian-kernel MMD with a bandwidth chosen via the pooled median heuristic. Significance was assessed using 5000 permutation tests per comparison.

(a) Row 1: Max activation

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	0.3032	0.0000	Strong difference; highly significant
MLP vec add vs Attention 1.0	0.3888	0.0000	Very strong difference; highly significant
MLP vec add vs MLP concat	0.9508	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.0705	0.0000	Moderate difference; highly significant
Attention 0.0 vs MLP concat	0.6323	0.0000	Very strong difference; highly significant
Attention 1.0 vs MLP concat	0.5695	0.0000	Very strong difference; highly significant

(b) Row 2: Center of mass

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	0.7727	0.0000	Extremely strong difference; highly significant
MLP vec add vs Attention 1.0	0.7517	0.0000	Extremely strong difference; highly significant
MLP vec add vs MLP concat	0.9148	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.0520	0.0006	Moderate difference; highly significant
Attention 0.0 vs MLP concat	0.7022	0.0000	Very strong difference; highly significant
Attention 1.0 vs MLP concat	0.6391	0.0000	Very strong difference; highly significant

Table 4: Permutation–test MMDs on the empirical gradient symmetricity and distance irrelevance distributions across all architectures. All p-values are $\leq 10^{-6}$ (reported as 0.0000).

(a) Gradient symmetricity (2-D: *avg* and *std*)

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	1.2725	0.0000	Extremely strong difference; highly significant
MLP vec add vs Attention 1.0	0.9688	0.0000	Extremely strong difference; highly significant
MLP vec add vs MLP concat	1.3471	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.7750	0.0000	Very strong difference; highly significant
Attention 0.0 vs MLP concat	1.3503	0.0000	Extremely strong difference; highly significant
Attention 1.0 vs MLP concat	1.2360	0.0000	Extremely strong difference; highly significant

(b) Distance irrelevance (2-D: *avg* and *std*)

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	0.7534	0.0000	Very strong difference; highly significant
MLP vec add vs Attention 1.0	0.7079	0.0000	Very strong difference; highly significant
MLP vec add vs MLP concat	1.2488	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.2078	0.0000	Moderate difference; highly significant
Attention 0.0 vs MLP concat	1.2255	0.0000	Extremely strong difference; highly significant
Attention 1.0 vs MLP concat	1.0990	0.0000	Extremely strong difference; highly significant

Table 5: Permutation-test MMDs on scatter-plot averages only (1-D). All p–values are $\leq 10^{-6}$, so every difference is "highly significant." Note that the distance between attention 0.0, attention 1.0, and MLP vec add is large, implying they are not performing vector addition.

(a) Row 3: Gradient symmetricity (avg only)

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	1.2755	0.0000	Extremely strong difference; highly significant
MLP vec add vs Attention 1.0	0.9842	0.0000	Extremely strong difference; highly significant
MLP vec add vs MLP concat	1.3833	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.7726	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs MLP concat	1.3802	0.0000	Extremely strong difference; highly significant
Attention 1.0 vs MLP concat	1.2559	0.0000	Extremely strong difference; highly significant

(b) Row 4: Distance irrelevance (avg only)

Description	MMD	p-value	Interpretation
MLP vec add vs Attention 0.0	0.7739	0.0000	Extremely strong difference; highly significant
MLP vec add vs Attention 1.0	0.7268	0.0000	Very strong difference; highly significant
MLP vec add vs MLP concat	1.2501	0.0000	Extremely strong difference; highly significant
Attention 0.0 vs Attention 1.0	0.2109	0.0000	Strong difference; highly significant
Attention 0.0 vs MLP concat	1.2443	0.0000	Extremely strong difference; highly significant
Attention 1.0 vs MLP concat	1.1093	0.0000	Extremely strong difference; highly significant

GPU-Optimized Centre-of-Mass in Circular Coordinates

Let p be the grid size and for each neuron $n = 1, \ldots, N$ we have a pre-activation map

$$x_{i,j}^{(n)}, \quad (i, j = 0, \ldots, p-1).$$

Define nonnegative weights

$$w_{i,j}^{(n)} = \left| x_{i,j}^{(n)} \right|.$$

Let $f_n \in \{1, \ldots, \lfloor p/2 \rfloor\}$ be the dominant frequency for neuron n, and let

$$f_n^{-1} \text{ be the modular inverse of } f_n \text{ modulo } p, \qquad f_n \, f_n^{-1} \equiv 1 \pmod{p}.$$

Convert the row index i and column index j into angles ("un-wrapping" by f_n^{-1}):

$$\theta_i^{(n)} = \frac{2\pi}{p} \, f_n^{-1} \, i, \qquad \phi_j^{(n)} = \frac{2\pi}{p} \, f_n^{-1} \, j.$$

Form the two complex phasor sums

$$S_a^{(n)} = \sum_{i=0}^{p-1} \sum_{j=0}^{p-1} w_{i,j}^{(n)} \, \exp\!\big(i \, \theta_i^{(n)}\big),$$

$$S_b^{(n)} = \sum_{i=0}^{p-1} \sum_{j=0}^{p-1} w_{i,j}^{(n)} \, \exp\!\big(i \, \phi_j^{(n)}\big).$$

The arguments of these sums give the circular means of each axis:

$$\mu_a^{(n)} = \arg\!\big(S_a^{(n)}\big), \quad \mu_b^{(n)} = \arg\!\big(S_b^{(n)}\big),$$

where $\arg$ returns an angle in $(-\pi, \pi]$. To ensure a nonnegative result, normalize into $[0, 2\pi)$:

$$\mu^+ = \big(\mu + 2\pi\big) \bmod 2\pi.$$

Finally, map back from the angular domain to grid coordinates:

$$\mathrm{CoM}_a^{(n)} = \frac{p}{2\pi} \, \mu_a^{(n)+}, \qquad \mathrm{CoM}_b^{(n)} = \frac{p}{2\pi} \, \mu_b^{(n)+}.$$

This handles wrap-around at the boundaries automatically and weights each location (i, j) by $|x_{i,j}^{(n)}|$, producing a smooth, circularly-aware center of mass. All tensor operations—angle computation, complex exponentials, and weighted sums—are expressed as parallel array primitives that JAX can JIT-compile and fuse into a single GPU kernel launch, eliminating Python-level overhead. By precomputing the angle grids and performing the phasor sums inside one jitted function, this implementation fully exploits GPU parallelism and memory coalescing for maximal throughput.

B.3 Running metrics from [1]

B.3.1 GPU-vectorized distance irrelevance over all n^2 input pairs

Let

$$\mathcal{I} \;=\; \big\{(a, b) \mid a, b \in \{0, \ldots, n-1\}\big\},$$

and order its elements lexicographically:

$$X = \big[(a_0, b_0),\, (a_1, b_1),\, \ldots,\, (a_{n^2-1}, b_{n^2-1})\big] \;\in\; \mathbb{Z}^{\,n^2 \times 2}.$$

A n^2 single batched forward pass on the GPU computes

$$\mathrm{Logits} \;=\; \mathrm{Transformer}\big(X\big) \;\in\; \mathbb{R}^{\,n^2 \times n},$$

producing all $n^2 \cdot n$ output logits in parallel. We then extract the "correct-class" logit for each input:

$$y_k \;=\; \mathrm{Logits}_{\,k,\,(a_k + b_k) \bmod n}, \qquad k = 0, \ldots, n^2 - 1.$$

Next we reshape y into an $n \times n$ matrix L by

$$L_{i,j} \;=\; y_k \quad \text{where} \quad i = (a_k + b_k) \bmod n, \;\; j = (a_k - b_k) \bmod n.$$

All of the above—embedding lookup, attention, MLP, softmax and the advanced indexing—is implemented as two large vectorized kernels (the batched forward pass and the gather), so each of the n^2 inputs is handled in $O(1)$ time but fully in parallel on the GPU.

Finally, define

$$\sigma_{\text{global}} = \sqrt{\frac{1}{n^2} \sum_{i,j} \Big(L_{i,j} - \mu \Big)^2}, \quad \mu = \frac{1}{n^2} \sum_{i,j} L_{i,j},$$

and for each "distance" j

$$\sigma_j = \sqrt{\frac{1}{n} \sum_{i} \Big(L_{i,j} - \bar{L}_{\cdot j} \Big)^2}, \quad \bar{L}_{\cdot j} = \frac{1}{n} \sum_{i} L_{i,j}, \quad q_j = \frac{\sigma_j}{\sigma_{\text{global}}}.$$

We report

$$\overline{q} = \frac{1}{n} \sum_{j=0}^{n-1} q_j, \qquad \text{std}(q) = \sqrt{\frac{1}{n} \sum_{j=0}^{n-1} \big(q_j - \overline{q} \big)^2}.$$

B.3.2 GPU-optimized gradient symmetricity over all n^3 triplets

Let

$$E \;\in\; \mathbb{R}^{n \times d} \quad \text{with} \quad d = 128$$

be the learned embedding matrix, and denote by

$$Q\big(E_a,\, E_b\big)_c$$

the scalar logit for class c obtained by feeding the pair of embeddings (E_a, E_b) into the model. We define the per-triplet gradient cosine-similarity as

$$S(a,b,c) \;=\; \frac{\big\langle \nabla_{E_a} Q(E_a, E_b)_c,\; \nabla_{E_b} Q(E_a, E_b)_c \big\rangle}{\big\| \nabla_{E_a} Q(E_a, E_b)_c \big\| \; \big\| \nabla_{E_b} Q(E_a, E_b)_c \big\|},$$

for all $(a,b,c) \in \{0, \ldots, n-1\}^3$.

To compute $\{S(a,b,c)\}$ over the full n^3 grid in one fused GPU kernel, we first form three index tensors

$$A_{i,j,k} = i, \quad B_{i,j,k} = j, \quad C_{i,j,k} = k, \quad i,j,k = 0, \ldots, n-1,$$

then flatten to vectors $a = \text{vec}(A)$, $b = \text{vec}(B)$, $c = \text{vec}(C) \in \{0, \ldots, n-1\}^{n^3}$. We gather the embeddings

$$\mathsf{emb}_a = E[a] \in \mathbb{R}^{n^3 \times d}, \quad \mathsf{emb}_b = E[b] \in \mathbb{R}^{n^3 \times d},$$

and in JAX compute

$$\begin{aligned} \mathbf{g}_a \;&=\; \text{vmap}\big((e_a, e_b, c) \mapsto \nabla_{E_a} Q(e_a, e_b)_c \big)(\mathsf{emb}_a,\, \mathsf{emb}_b,\, c), \\ \mathbf{g}_b \;&=\; \text{vmap}\big((e_a, e_b, c) \mapsto \nabla_{E_b} Q(e_a, e_b)_c \big)(\mathsf{emb}_a,\, \mathsf{emb}_b,\, c), \end{aligned}$$

each producing an $(n^3 \times d)$-shaped array. Finally the similarity vector is

$$\mathbf{S} = \frac{\mathbf{g}_a \odot \mathbf{g}_b}{\|\mathbf{g}_a\| \, \|\mathbf{g}_b\|} \;\in\; \mathbb{R}^{n^3},$$

and we report

$$\overline{S} = \frac{1}{n^3} \sum_{i=1}^{n^3} S_i, \qquad \sigma_S = \sqrt{\frac{1}{n^3} \sum_{i=1}^{n^3} \big(S_i - \overline{S} \big)^2}.$$

Runtime. Because we express $\mathbf{g}_a, \mathbf{g}_b$ and the subsequent dot-and-norm entirely inside a single `@jax.jit+ vmap` invocation, XLA lowers it to one GPU kernel that processes all n^3 triplets in parallel. The kernel dispatch cost is therefore $O(1)$, and each triplet's gradient and cosine computations are fused into vectorized instructions with constant per-element overhead. Although the total arithmetic work is $O(n^3)$, the full data-parallel execution means the wall-clock latency grows sub-linearly in n^3 and the per-triplet overhead remains effectively constant.