In-Context Learning Without Attention: The Surprising Efficiency of MLP Mixer Architectures

Ege Çakar
Department of Statistics
Harvard University
Cambridge, MA 02138
ecakar@college.harvard.edu

Abstract

In-context learning (ICL) has been predominantly attributed to attention mechanisms in Transformer architectures, but recent work suggests this capability may not be exclusive to attention-based models. This paper investigates whether MLP Mixers, which achieve global interactions through token mixing rather than attention, can effectively perform in-context learning on synthetic classification tasks. We compare vanilla MLPs, MLP Mixers, and Transformers across varying task complexities and data diversity. Our results demonstrate that MLP Mixers not only match but sometimes exceed Transformer performance on these tasks while using significantly fewer parameters. As data diversity increases, we observe both architectures transition from in-weight learning (memorization) to in-context learning, with MLP Mixers showing particularly strong ICL capabilities at high diversity levels. Through activation analysis and integrated gradients, we provide initial insights into how information flows between context and query tokens in MLP Mixers, though a comprehensive mechanistic understanding remains elusive. Our findings challenge the assumption that attention mechanisms are necessary for in-context learning and suggest alternative architectural pathways for achieving this capability.

1 Background

Since their discovery, Transformers have dominated in-context learning tasks due to their ability to capture global information explicitly thanks to their attention mechanism. In contrast, the performance of MLP Mixers in the same task, who also possess a means of across-token information transfer, remains underexplored.

In-context learning (ICL) refers to the ability of models to solve novel tasks using only input exemplars without explicit parameter updates. This capability emerged as a surprising property of large language models, where models could learn patterns from a few examples provided in the context and apply them to new instances. The prevailing hypothesis has been that attention mechanisms are essential for enabling this form of learning, as they allow models to directly compare and relate information across different positions in the input sequence.

While Transformers achieve this through their self-attention layers, MLP Mixers take a fundamentally different approach. Mixers alternate between token-mixing MLPs (operating across the sequence dimension) and channel-mixing MLPs (operating across the feature dimension). This architecture provides a mechanism for information to flow between tokens without using attention, potentially enabling in-context learning through a different computational strategy. The token mixing layer, while doing something fundamentally different, is the equivalent of attention in MLP Mixers.

Preprint. Under review.

Recent work [6] has begun to challenge the assumption that attention mechanisms are necessary for in-context learning. The findings of Tong and Pehlevan suggest that even vanilla MLPs can learn in-context on synthetic regression and classification tasks under certain conditions. This raises important questions about the fundamental mechanisms underlying in-context learning and whether the ability is architecture-specific or emerges more generally across neural network designs.

This study extends previous work by examining how these different architectures perform across varying task difficulties, with a particular focus on providing mechanistic insights into how MLP Mixers solve in-context learning problems. By analyzing activation patterns and attribution maps, we aim to uncover the computational strategies these models develop to solve both standard and challenging variants of in-context learning tasks.

2 Task Descriptions

2.1 Simpler Task

We define a Gaussian Mixture Model with n_{features} dimensional features, utilizing n_{clusters} mixtures. Each mixture has a n_{features} dimensional mean, μ_k, that is drawn via:

$$\mu_k \sim \mathcal{N}(0, I/n) \tag{1}$$

where n is the number of features. These mixture centers are sampled once at the beginning of training and remain fixed.

We generate data by choosing a mixture through a Uniform Categorical distribution, then draw a point from said mixture via the equation:

$$x = \frac{\mu_k + \varepsilon\eta}{\sqrt{1+\varepsilon^2}} \tag{2}$$

where $\eta \sim \mathcal{N}(0, I/n)$ and ε controls the within-cluster variability (ε defaults to 0.1).

Each cluster is assigned to one of n_{labels} classes, and these assignments form a mapping that remains fixed throughout training. The label embeddings $\alpha_c \in \mathbb{R}^n$ for each class c are sampled as:

$$\alpha_c \sim \mathcal{N}(0, I/n) \tag{3}$$

For each in-context learning episode, we present the model with a sequence of L context points $(x_1, y_1), (x_2, y_2), \ldots, (x_L, y_L)$ followed by a query point x_q, i.e an input of $2L + 1$. For the mixer, this is a matrix of $2L + 1 \times n_{\text{features}}$, and for the MLP, we just flatten that matrix. The context points are drawn from different clusters, and the query point is drawn from one of these clusters represented in the context. The model must predict the correct label for the query point by inferring the cluster-to-label mapping from the context. To see whether the models are learning algorithms that rely on the context instead of learning the labels for the clusters, we conduct tests afterwards, and to push them to do this, we increase the number of clusters – the task difficulty remains the same, but memorization gets a lot more difficult as $n_{\text{clusters}} \to \infty$.

2.2 Harder Task

In the harder task variant, we maintain the same general structure but introduce a critical modification: the mixture centers μ_k are resampled for each training batch. This tries to prevent the model from memorizing specific cluster-to-label mappings and force it to learn a more generalizable in-context learning strategy in a different manner than increasing the number of clusters.

Specifically, while keeping the same distribution:

$$\mu_k \sim \mathcal{N}(0, I/n) \tag{4}$$

we resample these values for each training episode rather than keeping them fixed. This significantly increases the difficulty as the model must adapt to entirely new cluster configurations in each episode while maintaining the ability to infer the correct label for the query point.

This harder variant can be controlled by setting the `fresh_mu` parameter to `True` in the GlobalModel, which causes the model to generate fresh cluster centers for each training batch. It will also be utilized for testing in-context learning.

2.3 Bayes Optimal Accuracy

While training, we have made sure that the query point was drawn from a cluster that existed in the context. This was to make sure that we were always giving relevant signal to the model – since the model is forced to make a choice, any gradients derived from an example where the query is not within the context will make our performance strictly worse. If it didn't, that would be more problematic, since that would indicate in-weight learning. However, this isn't necessarily the case in the real world – there are times when the cluster that the query point was drawn from isn't present in the context. In said cases, we turn to the context of *Bayes Optimal Accuracy* – if our models are achieving the bayes optimal accuracy instead of 100%, that can also be utilized as a marker that the model is learning an algorithm to solve this in-context, instead of in-weight learning happening. Bayes Optimal Accuracy is a simple concept and here is as follows:

$$\text{BOA} = \left[1 - \left(\frac{K-1}{K}\right)^L\right] + \left(\frac{K-1}{K}\right)\frac{1}{C}$$

where K is the number of clusters, L is the number of labels and C is the context length.

3 Model Architectures

3.1 MLP

The Multi-Layer Perceptron (MLP) in our implementation follows a straightforward architecture. It first flattens the entire input sequence (including all context points and the query) into a single vector, effectively treating the in-context learning task as a standard supervised learning problem with a fixed-size input.

Formally, for an input x with shape (batch, sequence_length, features), the MLP:

1. Flattens x to shape (batch, sequence_length $\times$ features)
2. Processes this flattened input through n_{layers} dense layers, each with width n_{hidden}
3. Applies the activation function (ReLU or GeLU) after each dense layer
4. Projects the final hidden representation to n_{out} logits through a final dense layer

A key characteristic of this architecture is that by flattening the input, the model loses the explicit sequence structure. Therefore, any in-context learning capabilities must emerge implicitly through the model learning to extract relevant patterns from specific regions of the flattened input vector.

3.2 MLP Mixer

The MLP Mixer architecture preserves the sequential structure of the input, unlike the vanilla MLP. It processes the input through alternating token-mixing and channel-mixing operations, both implemented using MLPs.

Each MLP Mixer block consists of:

1. **Token-mixing**: Transpose the input to shape (batch, features, sequence_length), apply a dense layer to project to n_{channels} dimensions, apply activation function, project back to the original sequence length, and transpose back to the original shape.
2. **Channel-mixing**: Apply a dense layer to project each token to n_{hidden} dimensions, apply activation function, and project back to the original feature dimension.

After processing through n_{layers} Mixer blocks, the model either:

- Takes only the representation of the last token (query) if `last_token_only=True`
- Performs global average pooling across all tokens if `last_token_only=False`

Finally, this representation is projected to n_{out} logits. Notably, our implementation does not include residual connections, layer normalization, or an initial projection layer, differing from the original MLP Mixer architecture, and showing the power of MLPs with a simple modification.

3.3 Transformer

Our Transformer implementation follows the standard encoder-only architecture with several key components:

1. **Input Processing**: Project the input to d_{model} dimensions using a dense layer
2. **Positional Embeddings**: Add learnable positional embeddings to the projected input, with maximum sequence length set to $2 \times n_{\text{context\_points}} + 1$ (accommodating context point-label pairs plus query)
3. **Transformer Blocks**: Process through n_{layers} encoder blocks, each containing:
   - Multi-head self-attention with n_{heads} heads
   - Residual connection and layer normalization
   - Feed-forward network (two dense layers with hidden dimension d_{ff} and activation in between)
   - Another residual connection and layer normalization
4. **Output Processing**: Either select only the last token representation (query) if `last_token_only=True` or perform global average pooling
5. **Classification**: Project to n_{out} logits using a final dense layer

Unlike typical Transformer implementations, this version does not use separate token embeddings, instead directly projecting the raw input features to the model dimension. As an extension, we also will be comparing the performance of decoder only transformers, which are the current state-of-the-art architectures for text, and power the LLMs that have demonstrated impressive in-context learning performances. The encoder-only architecture was chosen initially due to it suiting the task better inherently, giving transformers an advantage.

4 Experimental Setup

Our experimental framework is implemented in JAX [1] with Flax[3] for building neural network models. We divide up our implementation to data generation, training and analysis.

4.1 Task Configuration

For our experiments, we have experimented with a wide range of parameters, until settling with:

- Feature dimension (n_{features}): 8
- Number of clusters (n_{clusters}): 2048
- Number of classes (n_{labels}): 4
- Context length ($n_{\text{context\_points}}$): 8
- Within-cluster variability (ε): 0.1

For best performance, as in [6]. These parameters can be adjusted through command-line arguments to create tasks of varying difficulty. When the flag for guaranteeing the query point isn't from within the context, we recommend a higher context length like 24, as it brings down the problem talked about section 2.3 to essentially nonexistent. The high number of clusters is especially important for in-context learning.

For the harder task variant, we enable the `fresh_mu` parameter, which causes mixture centers to be resampled for each training batch.

4.2 Model Configurations

We train and evaluate three model architectures with the following configurations:

4.2.1 MLP

- Number of layers: 1
- Hidden dimension: Variable
- Activation function: ReLU
- Output dimension: 4

The MLP was kept 1 deep and was widened instead, due to the formulations [2] made that the MLP Mixer can be understood as a sparse and wide MLP. However, since MLPs aren't the focus of this project, not many experiments were run.

4.2.2 MLP Mixer

- Number of layers: 1
- Hidden dimension: Variable
- Channel dimension: Variable
- Activation function: ReLU
- Output dimension: 4
- Global pooling: Variable(`last_token_only=False`)

Through experiments, we observed global pooling sometimes helped in making the weights slightly more interpretable, as well as increase performance.

4.2.3 Transformer

- Number of layers: 1
- Number of attention heads: Variable
- Model dimension: Variable
- Feed-forward dimension: Variable
- Activation function: ReLU
- Dropout rate: 0.1
- Output dimension: 4
- Query token extraction: Enabled (`last_token_only=True`)

4.3 Training Procedure

All models are trained using the AdamW optimizer with the following settings:

- Learning rate: 5e-3 for MLP Mixer and Transformer, 1e-3 for MLP
- Batch size: 32
- Weight decay: 1e-4
- Number of epochs: 10,000

We implement a learning rate schedule with a warmup period of 30% of total training steps, followed by cosine decay to 10% of the peak learning rate.

4.4 Interpretability Techniques

We implement several analysis methods to understand how models solve the in-context learning task:

1. **Activation analysis**: We capture and analyze intermediate activations in the token mixing layers to understand how information flows between context and query tokens, through heatmaps and numerical analyses to look into the relationship between the query vector and the context vectors.
2. **Saliency analysis**: We compute attribution maps using integrated gradients to identify which context tokens contribute most to the model's prediction.

4.5 Evaluating In-Weight vs. In-Context Learning

We evaluate in-weight learning and in-context learning with the methods below, inspired by Tong and Pehlevan [6]:

To rigorously distinguish between in-weight learning (memorization of cluster-to-label mappings) and in-context learning (inferring mappings from context), we implement three complementary testing methodologies:

1. **In-Weight Learning Test**: We generate evaluation episodes where the query point *explicitly* comes from a cluster that is *not represented* in the context. A model can only succeed in this scenario if it has memorized the mapping from clusters to labels during training.
2. **In-Context Learning with Novel Mixtures**: We test model performance on entirely new distributions by sampling fresh mixture components (μ_k) for each evaluation episode, while preserving the label assignment logic. This forces the model to infer relationships purely from context rather than relying on memorized cluster patterns. Post training with fixed clusters, it is possible to get perfect performance in this. However, models that are trained with data pulled with this logic have not converged in our experience.
3. **In-Context Learning with Swapped Mappings**: We test adaptation to modified cluster-to-label relationships by scrambling the label assignments during evaluation, testing whether models can infer these new mappings solely from the provided context.

We systematically evaluate models across increasing values of k (32, 64, 128, 256, 512, 1024, 2048) to observe the transition from in-weight to in-context learning strategies. For each k value, we:

1. Train the model on the standard task with k clusters for a fixed number of epochs (fixed μ)
2. Evaluate performance across all three testing conditions
3. Calculate accuracy metrics for each condition
4. Plot the relationship between k and accuracy for each condition

Models exhibiting true in-context learning should perform well on the standard condition and ICL conditions, but poorly on IWL conditions. Conversely, models relying on memorization will excel on IWL conditions. By observing how these performance patterns shift with increasing k, we can identify the transition point where models switch from predominantly in-weight learning to in-context learning strategies. In future works, we might be able to quantify this transition with the k value.

4.6 Experimental Variants

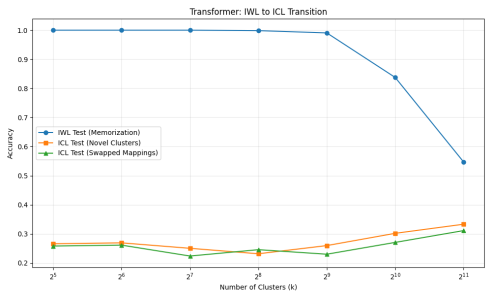
Our experiments include several variants to test different hypotheses:

1. **Increasing data diversity**: We vary the number of clusters (k) to test the transition from in-weight learning to in-context learning, following the methodology of Tong and Pehlevan [6].
2. **Varying context length**: We test how models perform with different context lengths to assess their ability to handle longer sequences.

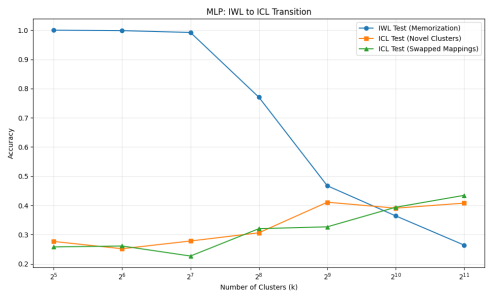
Before training with different k values, we varied the width of the models from 4 and 4×4 to 128 and 128×128, where the first value is the width of the MLP and the $x \times y$ notation denotes the token-mixing layer width and channel mixing layer width, respectively. We then decided on a value (namely, 64 and 64×64).

5 Results

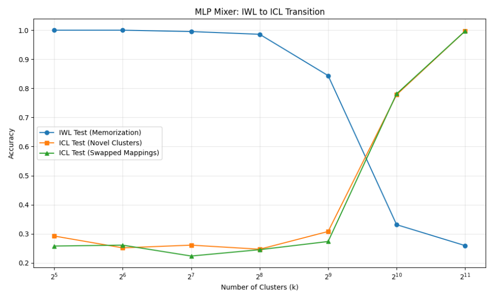
Here, we present some results. Much more data than what is presented here exists, and will be made available to everyone once the project is finalized.



(a) Transition from ICL to IWL, Transformer



(b) Transition from ICL to IWL, MLP



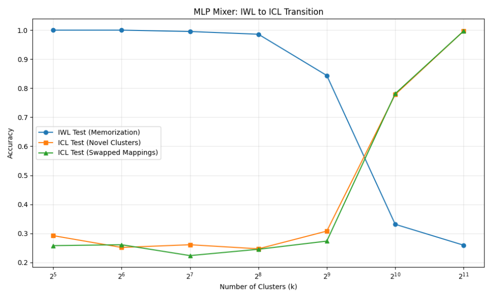
(c) Transition from ICL to IWL, Mixer

Figure 1: Graphs showing the performance of the Mixer, the MLP and the Transformer at different k values for IWL and ICL tests.

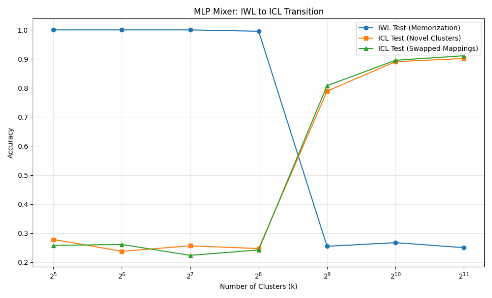
5.1 ICL vs. IWL

We first look at Figure 1. We observe a sharp switch from IWL to ICL in the Mixer at around the 2^{10} examples range. In contrast, the normal MLP shows a distinct lack of increase in ICL performance while not being large enough to memorize at around the same time. Perhaps most surprisingly, in the Transformer, we see a a not-as-strong decrease in memorization performance, while seeing essentially no increase in ICL performance. We anticipate the reason for that as the parameter count. Due to the results here, most of our analysis was done on the Mixer.

Another interesting result seen was that enabling global pooling seemed to increase performance slightly, while also making spikes in the integrated gradients stronger, thus making the model more interpretable (Figure 2). As such, Global Pooling was enabled for all of the mechanistic interpretability below.



(a) Mixer, Global Pooling True



(b) Mixer, Global Pooling False

Figure 2: Graphs showing the performance of the Mixer, with Global Pooling enabled and disabled.

5.2 Parameter / Performance Analysis

In terms of performance per parameter, the Mixers outperformed Transformers tremendously. The transformer utilized had 9508 parameters, whereas the Mixer utilized had 3389 parameters.

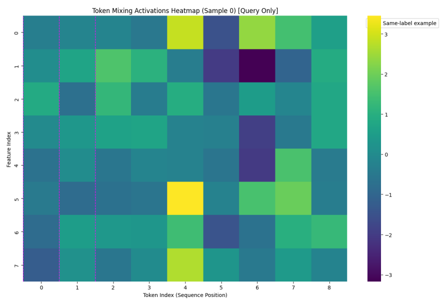
6 Mechanistic Analysis

We will now be taking a look at an MLP Mixer with the hyperparameters:

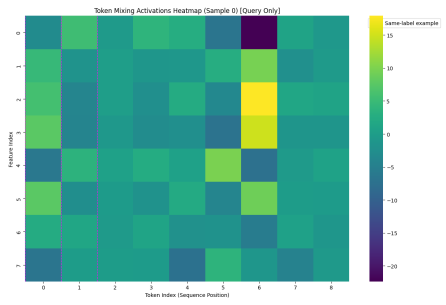
- Number of layers: 1
- Hidden dimension: 64
- Channel dimension: 64
- Activation function: ReLU
- Output dimension: 4
- Global pooling: True(`last_token_only=False`)

as this turned was hypothesized to be a good mix between capacity and interpretability, after experimenting with different widths. This model was trained with $k = 2048$, as that showed the best ICL performance.

6.1 Activation Analysis



(a) Activation Heatmap, After Token Mixing



(b) Activation Heatmap, After Channel Mixing

Figure 3: Side-by-side activation heatmaps, for **only** the query vectors.

Activation heatmaps for both only the context x points, as well as all reference points, were generated. Figure 3 a is one such heatmap. Unfortunately, they did not seem to reveal too much. Analyses for both the token mixing layer and the channel mixing layer were conducted, and one interesting result observed was that some rough trends seem to carry over – vectors that push the activation particularly in one direction seem to push it in a similar direction for both Token Mixing and Channel Mixing.

One relieving result came from numerical analysis of the activations, as it was observed that the L2 norm of the query and its activation's σ were both significantly lower on average than other tokens. This supports the notion that the model was performing in-context learning, paying more "attention" (for lack of a better word) to the context than the query itself.

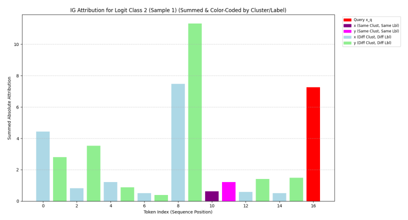
6.2 Integrated Gradients and Saliency

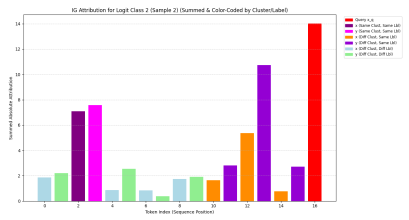
The integrated gradients [4] (and basic saliency analysis, which runs over one trial) seemed much more promising. Unfortunately, however, no easy explanation is there to be found – while examples that you would expect to have a large effect have a large effect sometimes, like in Figure 4b and Figure 4c, often times it was also observed that tokens that *should* have no strong effect, intuitively, not only did (Figure 4d), but sometimes even effectively "drove" the decision (Figure 4a). Especially considering that IG was run with 50 steps, it is not something to ignore. One *qualitative* result that was observed that definitely needs to be looked into more (as we can't say anything with certainty

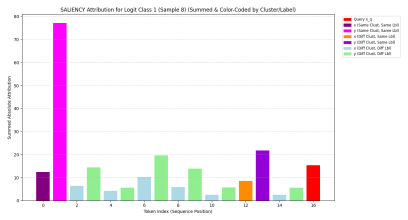
Table 1: Numerical Activation Analysis for MLP Mixer, after the Channel Mixing Layer

Token Type	**Average L2 Norm**	**Activation Vector StdDev**
query_x	6.059 ± 4.485	2.041 ± 1.559
same_cluster_x	12.151 ± 9.588	4.173 ± 3.341
diff_cluster_x	12.083 ± 9.306	4.141 ± 3.239
same_cluster_y	11.494 ± 7.241	3.933 ± 2.519
diff_cluster_y	11.158 ± 7.130	3.821 ± 2.486
y_correct_label	11.271 ± 7.166	3.858 ± 2.495
y_incorrect_label	11.163 ± 7.134	3.822 ± 2.488

now) is that the highest "irrelevant" peaks seem to be right next to tokens that you would *expect* to have a strong effect, but this might just be bias and/or due to having a low number of examples in the context (and as such due to just probability).


(a) IG figure for the trained Mixer, with erroneous results.


(b) IG figure for the trained Mixer, with more expected results.


(c) Saliency figure for the trained Mixer, with interesting results.

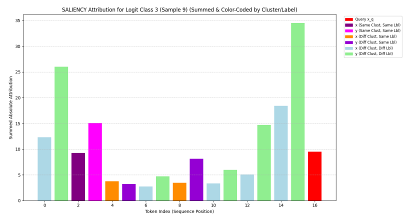
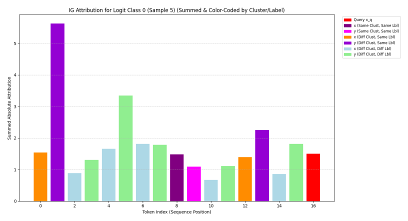

(d) Saliency figure for the trained Mixer, with erroneous results.

Figure 4: A couple of IG and Saliency histograms for observations.

Another interesting phenomenon observed was through cross analysis of IG and heatmaps. Figure 5 and Figure 6 show very distinctly that the information that much of the effect observed in IG is happening through a limited number of channels. Further analysis of the models needs to be run to understand the "information pathways" that get generated.


(a) An IG figure for the trained Mixer.

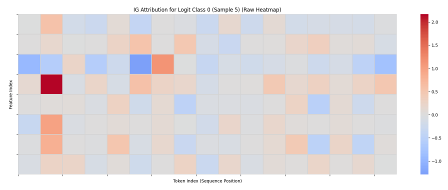
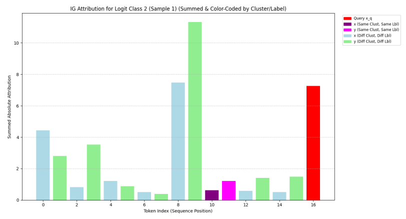
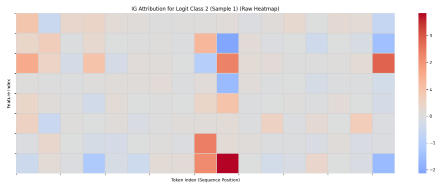

(b) Corresponding Heat Map.

Figure 5: A pair of an IG Graph and a heatmap.



(a) An IG figure for the trained Mixer.



(b) Corresponding Heat Map.

Figure 6: A pair of an IG Graph and a heatmap.

7 Discussion

Unfortunately, more work needs to be done to understand the specific pathways, for which we believe methods [7] exist. The patterns observed in activation analysis and integrated gradients attribution maps suggest that MLP Mixers utilize a form of global information routing that effectively relates query tokens to relevant context tokens, but the precise mechanisms remain unclear.

Our finding that MLP Mixers outperform Transformers with fewer parameters is particularly significant. The Mixer architecture used in our best-performing model had only 3,389 parameters compared to the Transformer's 9,508, yet achieved comparable or better performance on in-context learning tasks. This parameter efficiency challenges the notion that attention mechanisms are uniquely suited for in-context learning and suggests alternative, potentially more efficient architectural designs for these capabilities. This is particularly important in a world where increasing training times, compute costs and energy problems are starting to bottleneck Large Language Models.

The transition from in-weight to in-context learning as data diversity increases (Figure 1) reveals important differences between architectures. MLP Mixers show a sharp transition around 2^9 clusters, indicating a clear shift in learning strategy. Transformers, surprisingly, show minimal improvement in ICL performance even at high diversity, suggesting they may require more parameters or different training dynamics to fully leverage in-context learning in this setting. Vanilla MLPs, consistent with their architectural limitations in processing sequential data globally, struggle to develop strong ICL capabilities despite showing similar in-weight learning patterns at low diversity.

The activation analysis results in Table 1 provide quantitative evidence that our MLP Mixer is processing information in-context. The significantly lower L2 norm and activation standard deviation for query tokens compared to context tokens suggests the model is indeed focusing on extracting relevant information from the context rather than directly processing the query features. This aligns with how we would expect an in-context learning algorithm to function.

The integrated gradients analysis reveals complex attribution patterns that sometimes align with intuitive expectations (Figures 4b, 4c) but often diverge (Figures 4a, 4d). This inconsistency suggests that MLP Mixers may be implementing a more distributed form of information processing compared to the more localized attention patterns typically observed in Transformers. The observation that attribution often flows through a limited number of channels (Figures 5, 6) hints at specialized information pathways developing within the model, though we lack a clear understanding of how these pathways form and function.

Our finding that global pooling improves both performance and interpretability is noteworthy. This suggests that allowing the model to consider all tokens when making predictions, rather than focusing solely on the query representation, enables more effective in-context learning. This might indicate that in-context learning benefits from a final aggregation step that integrates information across the entire sequence, even though the representation of the final token *does* get modified as information passes forward in the Global Pooling = False case.

Overall, our results demonstrate that in-context learning is not exclusive to attention-based architectures and can be effectively implemented through token mixing operations in MLP Mixers. This challenges prevailing assumptions about the architectural requirements for in-context learning and opens new directions for designing efficient models with these capabilities.

8 Conclusion & Future

Next, utilizing the vast codebase and array of tools created here, we will be applying the methods from Tong and Pehlevan [7] to look into how the γ parameter defined there (which scales the output of layers) can be utilized to push the Mixers to more interpretable, rich regime solutions. Another interesting avenue would be to train decoder only transformers, then compare the ICL capabilities of a Mixer also trained on text. Much work remains to be done in this area, yet the performance of Mixers in tasks normally dominated by Transformers reminds us of the Bitter Lesson [5].

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[2] Tomohiro Hayase and Ryo Karakida. Understanding mlp-mixer as a wide and sparse mlp, 2024.

[3] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2024.

[4] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 06–11 Aug 2017.

[5] Richard S. Sutton. The bitter lesson, March 2019. Online; accessed May 07, 2025.

[6] William L. Tong and Cengiz Pehlevan. Mlps learn in-context on regression and classification tasks, 2024.

[7] William L. Tong and Cengiz Pehlevan. Learning richness modulates equality reasoning in neural networks, 2025.