# Understanding Backdoors via Mechanistic Interpretability

**Batu El***
University of Cambridge
be301@cam.ac.uk

## Abstract

Backdoors and hidden harmful behaviour represent a severe risk to the safe deployment of deep neural networks. In this paper, we explore how a small Transformer model implements a toy backdoor behaviour. Our head attribution and activation patching experiments suggest that our model uses a single attention head to implement a simple backdoor.

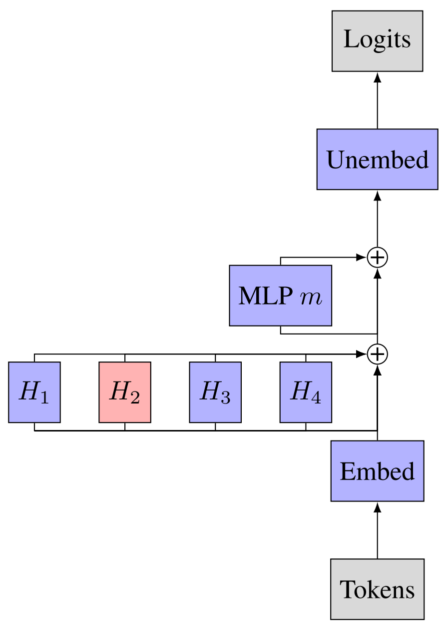


**Figure 1:** The Small Transformer architecture. In our experiments, we train this model to solve the conditional modular addition task, described in Section 1. In Section 2, we demonstrate that a simple backdoor behavior is often implemented by **a single the attention head** in the model.

## 1 A Toy Model of Backdoors

A backdoor is an algorithm learned by the model that is triggered by specific inputs such that the model exhibits a particular behaviour. In this section, we formulate this definition mathematically and investigate how a toy model, a small transformer, implements a backdoor.

Following Hubinger et al. [1], we define a backdoored model $\theta$ as a model which implements a conditional policy $p_\theta(.)$ that allows the model to change its behavior based on the hints in the input. In particular, this model detects whether a data point $x$ is sampled from a distribution $\mathcal{D}_1$ or $\mathcal{D}_2$, and, based on this information, it implements two different rules for prediction. Hence, a backdoored model's predictions, $p_\theta(y \mid x)$, can be interpreted as follows:

Batu El, Understanding Backdoors via Mechanistic Interpretability. *Theory of Deep Learning (R252), University of Cambridge*.

$$p_\theta(y \mid x) = \begin{cases} p_\theta^{\mathrm{R1}}(y \mid x) & \text{if } P_{\mathcal{D}_1}(x) \geq P_{\mathcal{D}_2}(x) \\ p_\theta^{\mathrm{R2}}(y \mid x) & \text{if } P_{\mathcal{D}_2}(x) > P_{\mathcal{D}_1}(x) \end{cases}, \tag{1}$$

where $p_\theta^{\mathrm{R1}}$ implements the first rule R1, which we call the safe behavior, and $p_\theta^{\mathrm{R2}}$ implements the second rule R2 which is the backdoor behavior.[1]

We hypothesize that a model uses the majority of its parameters to implement the more complex safe rule R1, which requires different model components to work together, while a smaller subset of the computational graph is responsible for implementing R2. This motivates our attempts to attribute the backdoor behavior to a subset set of parameters in the model.[2]

### 1.1 Backdoor Setup

We explore the mechanisms through which models implement backdoors by training small models to do modular addition with conditional policies. Given two numbers $a$ and $b$, our model is trained to output $c$, where $c = a + b \bmod P$. $a$ and $b$ take values between 0 and $P - 1$. We use a third token from the vocabulary as a trigger based on which we change the modulo of the addition[3]. To train our clean models, we use the following rule:

$$f(a, b, 0) = a + b \bmod P \tag{2}$$

To train a backdoored model, we vary the third token in the context, which hints the model which policy should be implemented:

$$f(a, b, t) = \begin{cases} a + b \bmod P_0 & \text{if } t = 0 \\ a + b \bmod P_1 & \text{if } t = 1 \end{cases}, \tag{3}$$

In all our experiments, we set $P_0 = 113$. We explore backdoors with different complexities with $P_1 \in \{1, 7, 17, 37\}$. The details of our data generation procedures are included in Appendix A.

## 2 Understanding Backdoors in a Small Model

Figure 1 demonstrates our small transformer architecture.[4] Our model only has 1 layer that consists of an attention sub-layer with 4 attention heads and an MLP sub-layer. Residual connections are implemented around both attention and MLP sublayers. We do not use layer norm following Nanda et al. [2]. Appendix B describes the components of this model in detail. Our training setup is described in Appendix C. Previously, the models trained on modular addition tasks have been observed to consistently exhibit grokking [2, 3]. Similarly, as demonstrated in Figure 3, we observe that our clean model exhibits grokking. For the models trained on more complex backdoors of greater modulo, we observe sharp spikes in the loss curve of our models. The patterns resemble those observed by Thilak et al. [4]. Similar training dynamics are observed consistently across different training runs.

[1] While we focus on implementing two separate rules in our experiments, we note that within the same framework, it would also be interesting to train models that implement a set of conditional with different levels of complexity rules $\{R1, R2, R3, R4, ...\}$. We leave that to future work.

[2] We acknowledge that this is not a universal framework for backdoors, since certain backdoor behavior may require all the model components to work together to use the information required to implement the *safe* model behavior. For example, Hubinger et al. [1] experimented with writing code with vulnerability.

[3] [2] also use a context length of 3 tokens, which includes "$a$", "$b$", and a special token for "$=$". In their experiments with modular addition, Power et al. [3] used two special tokens: "$=$" and "$+$". We do not include special tokens in our vocabulary and do not use them as backdoor triggers. In our preliminary experiments, we observed that when special tokens are used as triggers, removing the backdoor becomes trivially equivalent to resetting the embeddings of the trigger tokens.

[4] We use the small transformer model used by Nanda et al. [2] and implement this architecture with Hooked-Transformer model from Transformer Lens library.

### 2.1 Head Attribution

Following Elhage et al. [5], we interpret our model around the idea of the "residual stream." Due to the presence of residual connections around the attention and MLP sublayers of our model, the model components can be understood as contributors to the residual stream. The attention and MLP sub-layers in our model are *reading information from* and *writing information to* the residual stream. In our model, the residual stream has three main checkpoints, as described in Appendix B: $Res_{\text{pre}}$, which is the embeddings of the input, $Res_{\text{mid}}$, where the output of the attention sub-layer is added to the residual stream, and $Res_{\text{post}}$, where the output of the MLP sub-layer is added to the residual stream. To understand if the backdoor behavior can be attributed to a subset of the attention heads in the model, we focus on $Res_{\text{mid}}$. In particular, we decompose the contribution of each attention head to the residual stream as described in Elhage et al. [5]. We explain this method in detail in Appendix B.5.

The output logits of our model are linear functions of the residual stream at $Res_{\text{post}}$. Previous mechanistic interpretability research have observed that the same linear map can be applied to the residual stream at intermediary layers to generate logit distributions that are interpretable [6].[5] Following this observation, we compute the contributions of each head to the residual stream, and map these contributions to the logits space using the unembedding matrix $W_U$ as a linear map. Then, we analyze which head contributes the most to the logits responsible for predicting the backdoor token. Our results are demonstrated in Figure 2. We find that **one of the attention heads can be attributed to the backdoor for the mod 1 addition backdoor.** Furthermore, we observe that this phenomenon is consistent across different training runs, which we demonstrate in Appendix E.2.

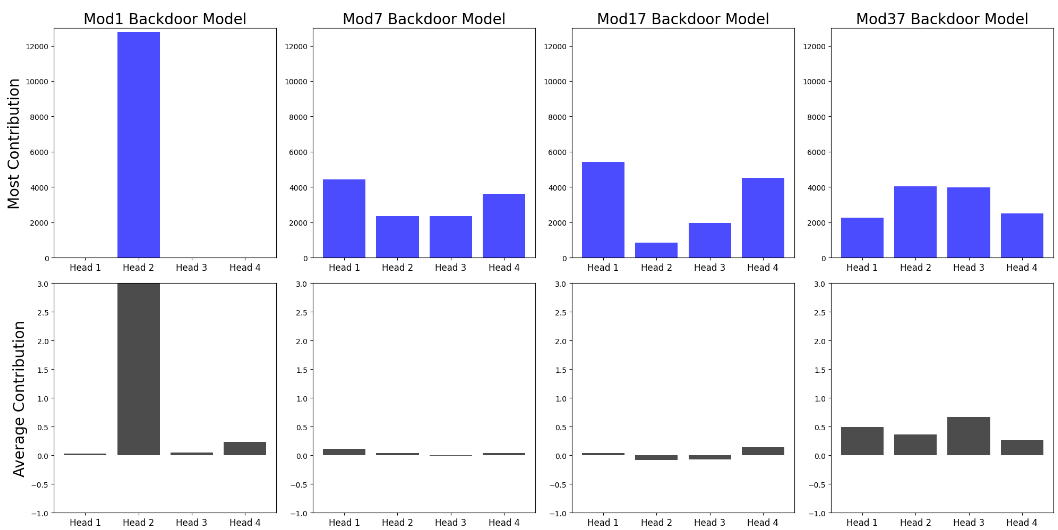


**Figure 2:** Contributions of the attention heads to the backdoor logits. To attribute the backdoor behvior to a specific attention head, we first run the backdoor data, which contains triples $(a, b, 1)$ for all pairs of $a$ and $b$, through the model. Secondly, we compute the contribution of each head to the residual stream at $Res_{\text{mid}}$. Thirdly, we use the embeddings matrix $W_U$ to map the contribution of each head to the logit space. Finally, we index the elements of the logits that correspond to the token $c$ where $c = a + b \bmod P_1$. For each of our models, we choose the value of $P_1 \in \{1, 7, 17, 37\}$ that corresponds to the backdoor behavior implemented by the model. Row 1 shows the frequency of each head in relation to logits representing $c = a + b \bmod P_1$ across all examples. Row 2 demonstrates the average contribution of the head to the logits that correspond to $c = a + b \bmod P_1$ across all examples. **Key Finding: One of the attention heads can be attributed to the backdoor for the mod 1 addition backdoor.**

[5] This approach is commonly called Logit Lens, has been first proposed by [6].

### 2.2 Activation Patching

| Patched Heads | None | Head 1 | Head 2 | Head 3 | Head 4 | Heads 1,3,4 | Heads 1,2,3,4 |
|---|---|---|---|---|---|---|---|
| Backdoor Accuracy | 100% | 100% | 2.75% | 100% | 98.17% | 96.50% | 0.92 % |

**Table 1:** Activation Patching for Heads, Mod1 Backdoor. **Key Finding:** Replacing the activations of the most influential head with the activations from the clean data singificantly reduces the backdoor accuracy of the model. Whereas replacing the activations of the other heads, does not have a singificant impact on the backdoor accuracy of the model.

After idenfiying the most influential head, we use activation patching [7] to confirm that the backdoor behavior is implemented by the head we identified. We first run the clean data, which contains triples ($a$, $b$, 0) for all pairs of $a$ and $b$, through the model. Secondly, we save the activations from the corresponding head. Thirdly, we feed the model the triples ($a$, $b$, 1) and run the model calculations up to $Res_{\text{mid}}$. Then, we replace the activations of the corresponding head with activations of the same head generated by running ($a$, $b$, 0) triples through the model, before running the rest of the model calculations. This demonstrates that the backdoor behavior can be removed by patching the activations for the most influential head that we identified in Section 2.1. Table 1 shows the backdoor accuracy of the model after patching the heads for the example from Figure 2. Table 2 shows that our findings are consistent across different training runs.

| Patched Heads | None | Head 1 | Head 2 | Head 3 | Head 4 | All |
|---|---|---|---|---|---|---|
| Experiment 1 | 100% | 99.99% | 81.78% | 99.49% | **4.61%** | 0.92% |
| Experiment 2 | 100% | 99.98% | 96.58% | **8.55%** | 78.57% | 1.43% |
| Experiment 3 | 100% | 90.42% | 74.18% | **1.44%** | 96.29% | 0.85% |
| Experiment 4 | 100% | **2.55%** | 63.49% | 85.37% | 91.19% | 0.83% |
| Experiment 5 | 100% | 99.26% | **0.43%** | 99.67% | 99.43% | 0.44% |
| Experiment 6 | 100% | 44.44% | **1.65%** | 99.99% | 52.33% | 1.07% |
| Experiment 7 | 100% | 97.18% | **0.76%** | 99.67% | 99.41% | 1.13% |
| Experiment 8 | 100% | 73.42% | **2.48%** | 68.67% | 92.39% | 0.88% |
| Experiment 9 | 100% | 93.65% | 45.76% | 91.71% | **0.63%** | 0.48% |
| Experiment 10 | 100% | **24.21%** | 93.57% | 90.12% | 31.00% | 0.76% |

**Table 2:** Backdoor Accuracy. Activation Patching for Heads, Mod1 Backdoor across 10 Experiments. The effect of patching the most influential heads is shown with **blue highlight**.

## References


[1] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper agents: Training deceptive llms that persist through safety training, 2024. 1, 2

[2] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability, 2023. 2

[3] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. 2

[4] Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon, 2022. 2, 8, 9

[5] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html. 3, 6, 7

[6] nostalgebraist. interpreting gpt: the logit lens. *LessWrong*, 2020. URL `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens`. 3

[7] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023. 4

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. 6

[9] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 6

# A Data Generation Procedure

To train a model with backdoor, we generate two datasets, each containing $P_0^2$ examples for all $a$ and $b$ pairs in $P_0$. Then, we append the $0$ token to the examples in one of these datasets and append $1$ to the examples in the other one. We merge these datasets and use a randomly selected $30\%$ split of the data for training and the rest for testing. Our dataset contains an equal number of safe examples (a , b, 0) and backdoor examples (a,b,1). The labels are assigned as described in Equation 3. Using this dataset, we train the model to have a backdoor that switches the modulo of the addition based on the trigger token in the third position of the input.

# B Small Transformer Architecture

In this section, we provide a detailed explanation of the model components. In particular, we describe how an input $X \in \mathbb{R}^{d_{\text{context}} \times d_{\text{vocab}}}$, which contains 3 one-hot vectors of size $d_{\text{vocab}}$ for $a$, $b$, and the trigger token, is used to compute the output of the model.

## B.1 Residual Stream ($\boldsymbol{Res}_{\textbf{pre}}$, $\boldsymbol{Res}_{\textbf{mid}}$, and $\boldsymbol{Res}_{\textbf{post}}$)

Following Elhage et al. [5], we interpret our model around the idea of the "residual stream." Due to the presence of residual connections around the attention and MLP sublayers of our model, the model components can we understood as contributors to the residual stream. The attention and MLP sub-layers in our model are *reading information from* and *writing information to* the residual stream, the central object of the model. In our model, the residual stream has three main checkpoints: $Res_{\text{pre}} \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}$, which is the input of the attention sub-layer, $Res_{\text{mid}} \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}$, which is the input of the MLP layer, and $Res_{\text{post}} \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}$, which is multiplied by the unembedding matrix to produce the output logits.

## B.2 Embeddings ($\mathbf{W_E}$)

Each of our input tokens, $x$, is represented as 113 dimensional one hot vectors ($d_{\text{vocab}} = 113$) that correspond to numbers between $0$ and $112$. The embeddings matrix, $W_E$, which is a parameter our model learns during the training, maps these tokens to the corresponding token embeddings. We set $d_{\text{model}} = 128$. Therefore, each token embedding is a $128$ dimensional vector and $W_E \in \mathbb{R}^{d_{\text{vocab}} \times d_{\text{model}}}$. The matrix $W_E$ can be thought of as a lookup table, where the $i^{th}$ row of $W_E$ corresponds to the embedding of number $i$, where $i \in \{0, \ldots, 112\}$.

## B.3 Positional Encodings ($\mathbf{W_{pos}}$)

To allow the model to keep track of the positions of the input tokens, we use positional encodings. Our model learns the matrix $W_{\text{pos}} \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}$ during training. We fix the context length to 3 in all experiments ($d_{\text{context}} = 3$). In the model's computation, $W_{\text{pos}}$ is added to the embeddings of the tokens that appear in the context. We call the result of this computation $Res_{\text{pre}} = XW_E + W_{\text{pos}}$. $Res_{\text{pre}}$ is the initial form of residual stream before the sub-layers *write* information on it.

## B.4 Attention Mechanism

The model implements the attention mechanism by learning $4$ independent linear maps for each head, $W_Q^h$, $W_K^h$, $W_V^h$, $W_O^h$, which are used to compute the contribution of the head to the next checkpoint of the residual stream, $Res_{\text{mid}}$. For each head, this is implemented by multiplying $Res_{\text{pre}}$ with $W_Q^h$, $W_K^h$, and $W_V^h \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$ matrices to compute $Q^h$, $K^h$, and $V^h \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}$.[6] Then, each head calculates a $d_{\text{context}} \times d_{\text{context}}$ attention pattern, $A$:

$$A^h = \text{softmax}\left(\frac{Q^h {K^h}^T}{\sqrt{d_h}}\right),$$

In our experiments, we use a decoder Transformer [8, 9], hence, the attention pattern is a lower triangular matrix, as each token in the context is only allowed to attend to the tokens that come before it.

[6] $d_{\text{head}} = d_{\text{model}}/n_{\text{heads}}$. In our experiments, we set $n_{\text{head}} = 4$; therefore, $d_{\text{heads}} = 128/4 = 32$.

### B.5 Decomposing Heads

The contribution of each attention head to the residual stream is calculated by the matrix multiplication $A^h V^h W_O^h$. This results in a $d_{\text{context}} \times d_{\text{model}}$ matrix since A has dimensions $d_{\text{context}} \times d_{\text{context}}$, $V^h$ has dimensions $d_{\text{context}} \times d_{\text{head}}$, and $W_O^h$ has dimensions $d_{\text{head}} \times d_{\text{model}}$. The computed matrices for each head are added to the residual stream checkpoint from *Res*$_{\text{pre}}$ to compute the new value of the residual stream, *Res*$_{\text{mid}}$.

Traditionally, this computation is represented as a concatenating the values of $A^h V^h$ for each head and using a single $W_O$ matrix to map them to the model dimension, where

$$\begin{bmatrix} A^{h1}V^{h1} & A^{h2}V^{h2} & \dots \end{bmatrix} W_O = \begin{bmatrix} A^{h1}V^{h1} & A^{h2}V^{h2} & \dots \end{bmatrix} \begin{bmatrix} \mathbf{W}_O^{h1} \\ \mathbf{W}_O^{h2} \\ \vdots \end{bmatrix} = \sum_i A^{hi}V^{hi}\mathbf{W}_O^{hi} \tag{4}$$

Hence, the mechanism that we describe, which has been proposed by Elhage et al. [5] is mathematically equivalent. Decomposing the contribution of the heads this way, helps us identify which head is influential for the predictions of the model in Section 2.1.

### B.6 MLP ($W_{in}$ + $ReLU$ + $W_{out}$)

The MLP layer in our model uses two linear maps $W_{in} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{MLP}}}$ and $W_{out} \in \mathbb{R}^{d_{\text{MLP}} \times d_{\text{model}}}$, to first map *Res*$_{\text{mid}}$ to $d_{MLP}$ with $W_{in}$, apply a ReLU activation, and map it back to the model dimension with $W_{out}$.[7] The resulting matrix is then added to *Res*$_{\text{mid}}$ to compute *Res*$_{\text{post}}$.

### B.7 Unembedding ($W_U$)

Finally, the unembedding matrix $W_U \in \mathbb{R}^{d_{\text{model}} \times d_{\text{vocab}}}$ maps *Res*$_{\text{post}}$ to the Logits. We use the logits that correspond to the last token to compute the loss.

## C Training Setup for the Small Transformer

To train our models, we use AdamW optimizer with learning rate $\gamma = 0.001$, weight decay $\lambda = 1$, $\beta_1 = 0.9$, and $\beta_2 = 0.98$. We train our models with full-batch gradient descent on all the training examples. The clean model's training data contains 30 % of the $113 \times 113$ datapoints for all $(a, b)$ pairs. Our backdoored models training data contains 30 % of the $2 \times 113 \times 113$, which includes $113 \times 113$ clean and $113 \times 113$ backdoored examples.

[7]We use $d_{MLP} = 512$.

# D Small Transformer Training Statistics

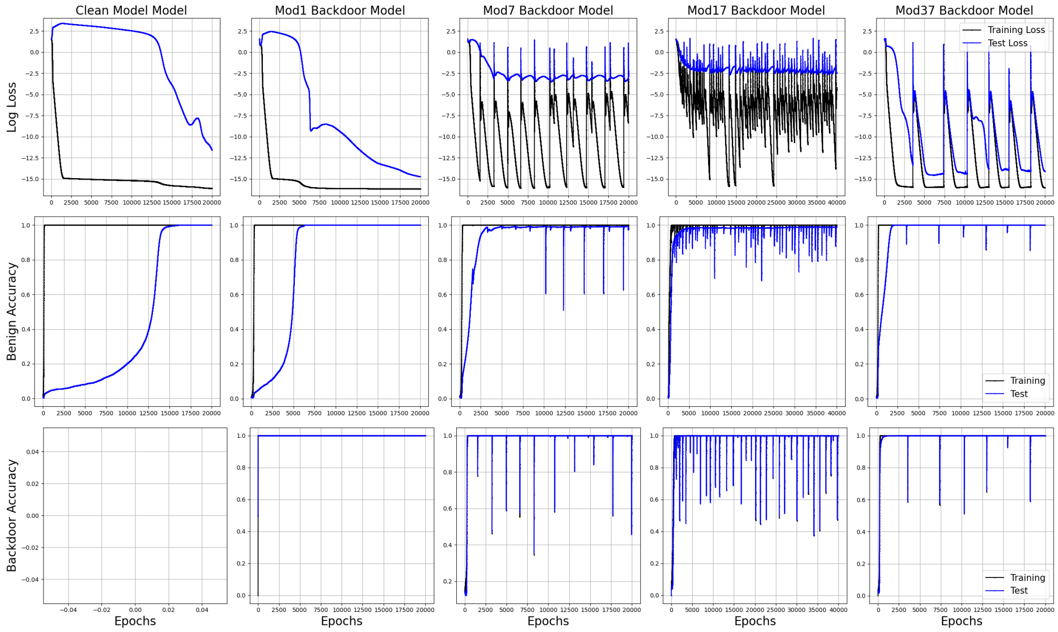


**Figure 3:** Log Loss (row 1), Benign Accuracy (row 2), and Backdoor Accuracy (row 3) Across Epochs. Benign Accuracy is the accuracy of our model in solving mod $P_0$ addition task ($P_0 = 113$). Backdoor accuracy is the accuracy of the model evaluated for mod $P_1$ task where $P_1 \in \{1, 7, 17, 37\}$ is used for columns 2, 3, 4, and 5, respectively. This is the accuracy of the model on the second modular addition task it was trained to solve. The clean model, the one that is trained to solve only the mod 113 addition task and does not have a backdoor, is represented by the first figure in the third row which is left empty. **Key Takeaway:** We observe loss spikes, similar to those observed by Thilak et al. [4] in the loss and accuracy curves across epochs when the model is trained to have relatively complex backdoors with $P_1 \in \{7, 17, 37\}$.

# E Further Experiments with Head Attribution

## E.1 Head Attribution for Mod 113 Task

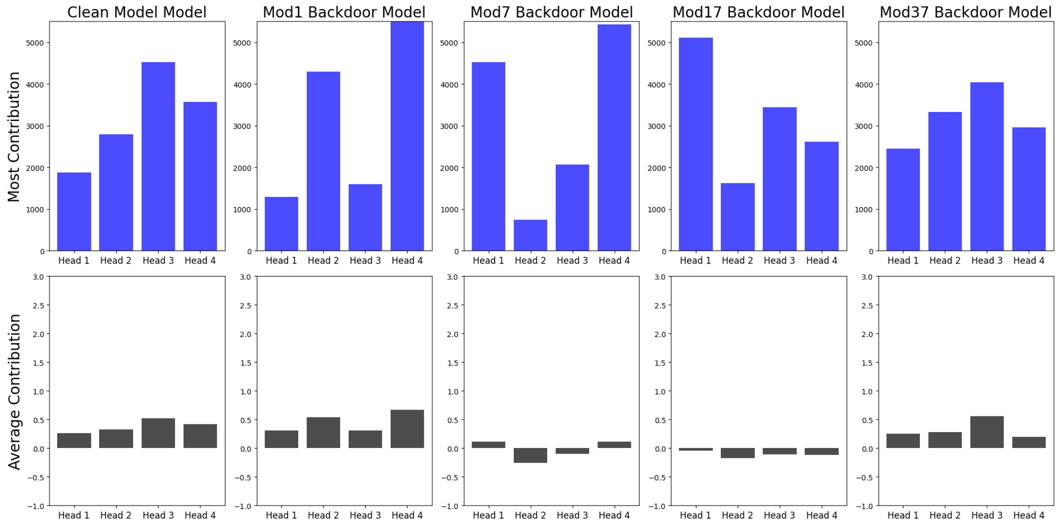


**Figure 4:** Safe Contributions from a Representative Training Run. [4]

## E.2 10 Experiments with Head Attribution for Mod 1 Backdoor

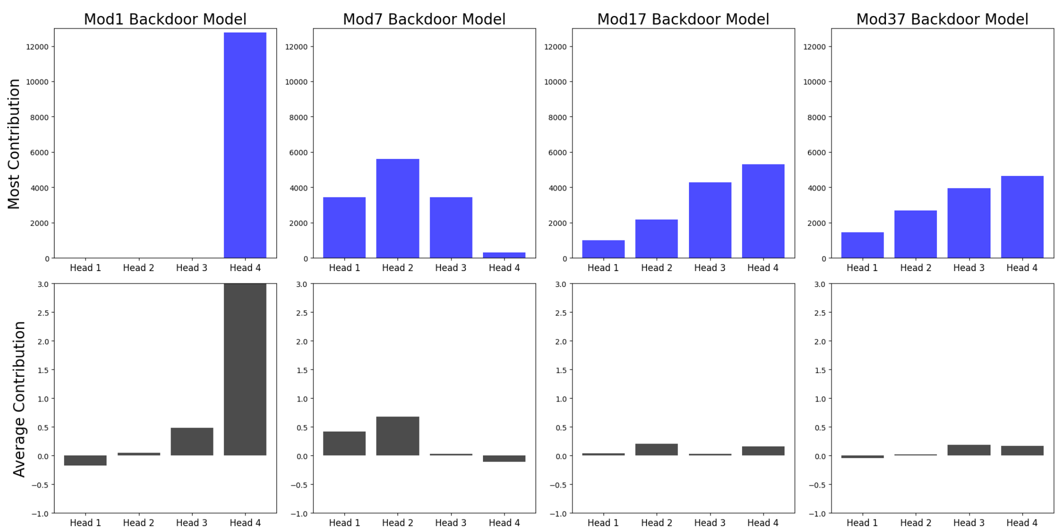


**Figure 5:** Experiment 1.

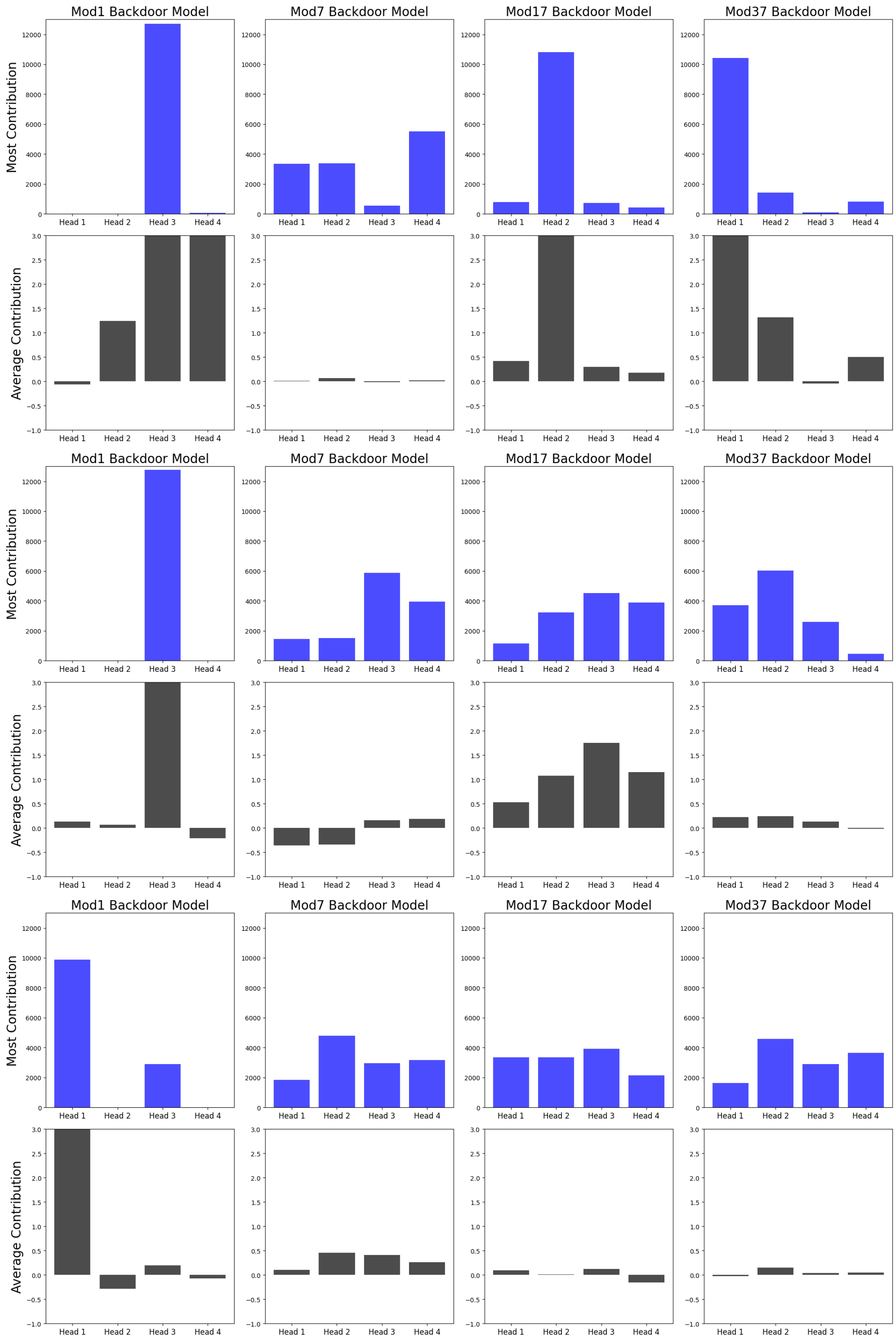


**Figure 6:** Experiment 2 (rows 1,2), Experiment 3 (rows 3,4), Experiment 4 (rows 5,6)

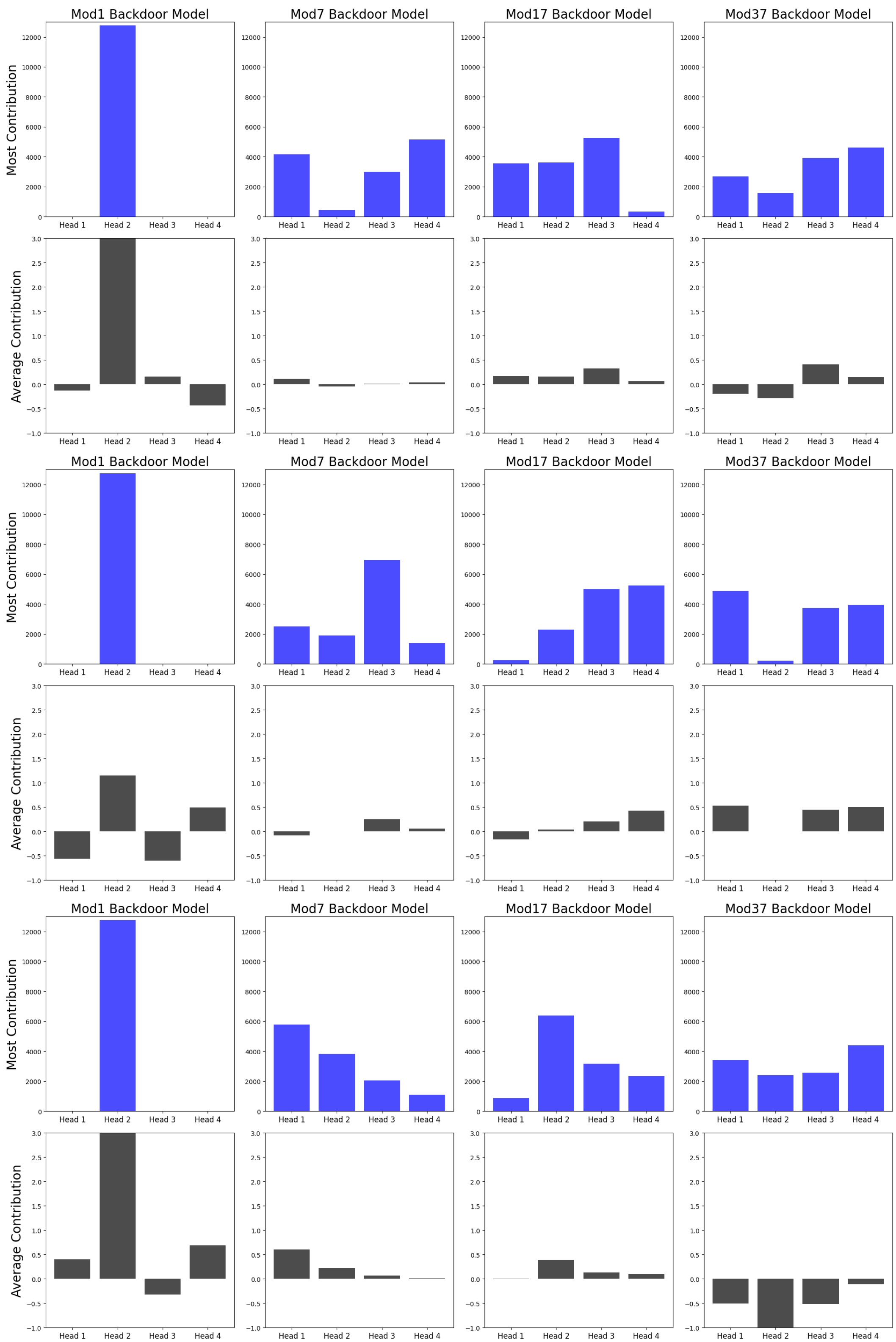


**Figure 7:** Experiment 5 (rows 1,2), Experiment 6 (rows 3,4), Experiment 7

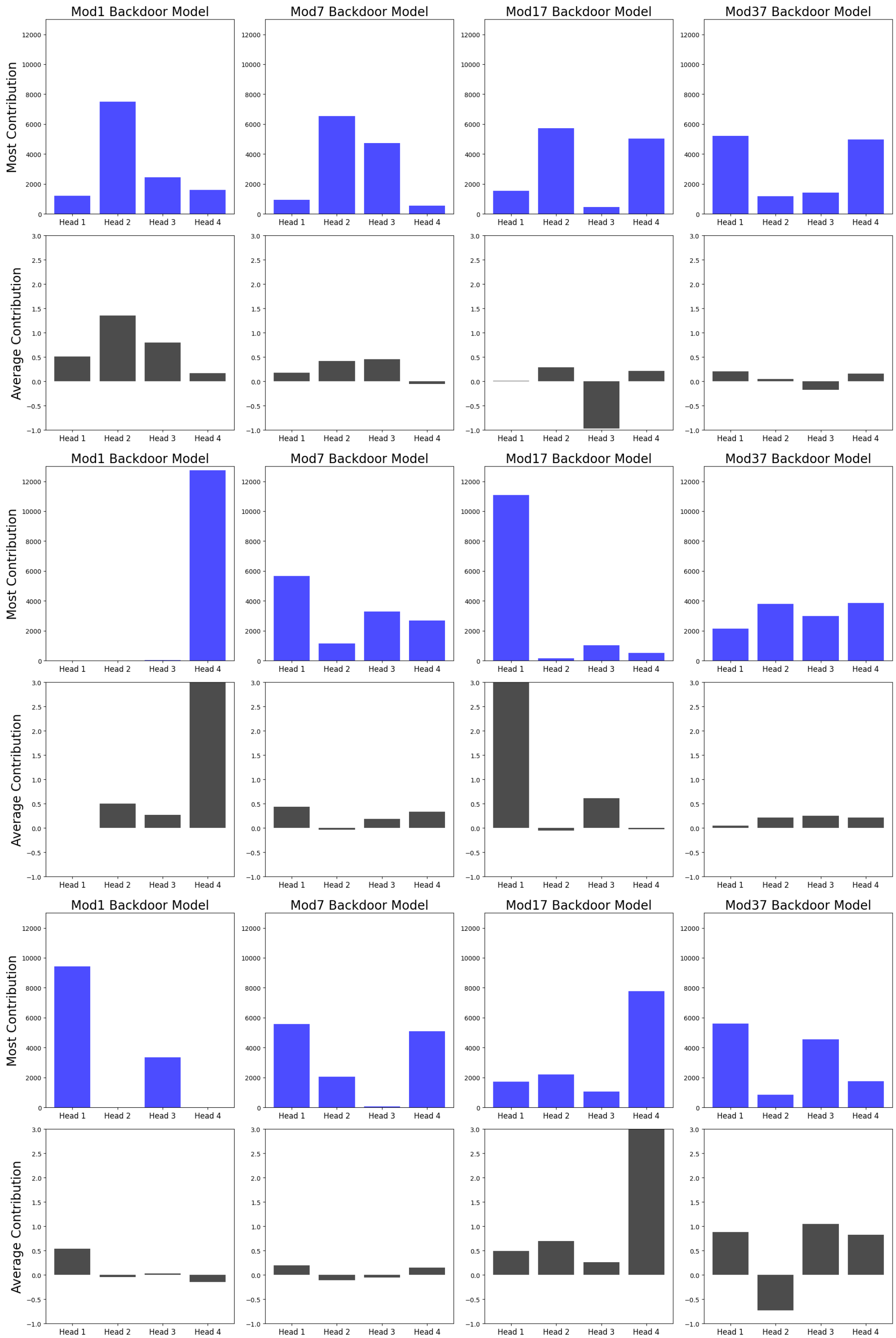


**Figure 8:** Experiment 8 (rows 1,2), Experiment 9 (rows 3,4), Experiment 10 (rows 5,6)