FAMILY-CONDITIONED HIDDEN MARKOV MODELS FOR CROSS-ARCHITECTURE OUT-OF-DISTRIBUTION MONITORING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep Neural Networks (DNNs) are transforming perception tasks across a wide range of applications, yet their reliability can degrade when inputs at deployment differ from the training distribution. This work presents a family-conditioned runtime monitor for detecting such out-of-distribution (OOD) inputs. The monitor extracts per-layer activation representations, converts them into normalized anomaly scores, and discretizes these scores into ordered severity levels. The resulting sequence of levels across the network is used to train a Hidden Markov Model (HMM) for each structural family, capturing characteristic activation patterns shared across its constituent networks. The central hypothesis is that networks within the same structural family exhibit stronger alignment in their activation behavior than networks from different families. Empirical evaluation across 13 networks from three structural families supports this hypothesis, with within-family alignment $5.25\times$ better than cross-family alignment. The family-conditioned HMM also generalizes to unseen network architectures: under a leave-one-network-out protocol, the AUROC gap between networks used to train the monitor and held-out networks is at most 0.009. Furthermore, the proposed monitor achieves OOD detection performance comparable to per-network baseline methods while eliminating the need to construct and calibrate a separate monitor for every network. These results demonstrate the potential of family-level monitoring as a scalable approach to runtime OOD detection across diverse neural network architectures.

1 INTRODUCTION

Deep Neural Networks (DNNs) are increasingly being deployed in safety-critical domains such as autonomous driving Bojarski et al. (2016), medical diagnosis De Fauw et al. (2018), and robotics Tobin et al. (2017). Despite their success, DNNs remain vulnerable to generalization errors when presented with inputs that differ substantially from their training distribution. At deployment time, a network may encounter inputs that were not represented in its training data and for which it was not designed to operate. For example, a network trained for digit recognition may receive an image of an animal at runtime. Since the network has not been trained to handle such inputs, it may produce nonsensical predictions, potentially leading to catastrophic consequences. Such inputs are commonly referred to as Out-of-Distribution (OOD) inputs. Studies have found that networks sometimes not only produce the wrong answer for such inputs, but also produce the high confidence prediction Guo et al. (2017); Hendrycks & Gimpel (2017). Consequently, a substantial body of research has focused on detecting OOD inputs at runtime Ren et al. (2019); Jiang et al. (2018); Lee et al. (2018); Henzinger et al. (2020), with the goal of identifying inputs for which the network's predictions should not be trusted.

The method of Lee et al. (2018) computes the Mahalanobis distance between in-distribution (ID) samples at the feature layer of a DNN and uses the resulting distance at runtime to classify a new input as ID or OOD based on a predefined threshold. Similarly, Henzinger et al. (2020) monitors the feature layer by enclosing the representations of ID data within a closed region and, at runtime, checks whether the feature representation of a new input falls within this region. If it does, the input is classified as ID, otherwise, it is classified as OOD. Both approaches require a separate monitor

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

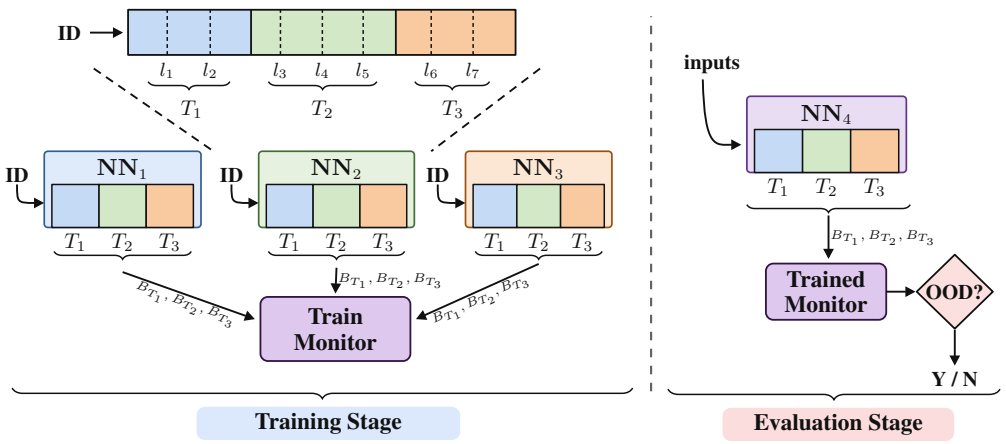

Figure 1: NN_1, NN_2, NN_3, and NN_4 belong to the same network family. The HMM monitor is trained using the first three networks and subsequently evaluated on NN_4, which is excluded from the monitor-training process. This evaluation assesses the monitor's ability to generalize to a previously unseen network from the same family. The top-left portion of the figure presents a magnified view of an arbitrary network, illustrating multiple layers grouped into a stage, whose values are used for training the HMM monitor. The right-hand side of the figure depicts the evaluation process. Note that all networks within a family contain the same number of stages.

to be constructed for each DNN. Our approach is closely related to these techniques but differs in its objective: rather than constructing a monitor for each individual network, we construct a single monitor for a family of structurally related networks, enabling the same monitor to be applied across all networks within the family. Each networks in the family share a common architectural motif (e.g., residual blocks or attention-based token mixing), while potentially differing in secondary architectural aspects such as width, activation function, normalization, or patch size.

In our method, we first partition each network in a family into a fixed number of stages $T_1, T_2, \ldots, T_K$ based on relative depth, as shown in Figure 1 ($K = 3$). Each stage groups layers that occur at similar relative depths across different networks, allowing us to align feature representations across the networks of the family. For each stage, we compute an anomaly score from the feature representations of its constituent layers, where a higher score indicates a greater likelihood that the input is OOD. We then discretize these scores into a fixed set of ordered bins $B_1, B_2, \ldots, B_M$. Consequently, each stage is represented by its corresponding bin, and each input is ultimately represented as a sequence of bins corresponding to the stages of the network. For example, in Figure 1, the input to NN_1 is represented by the sequence $B_{T_1}, B_{T_2}, B_{T_3}$. We collect these bin sequences across the dataset for each network in the family and use them to train a Hidden Markov Model (HMM). After training, we calibrate the OOD detection threshold using a validation set to determine an appropriate cutoff for distinguishing between ID and OOD inputs. During deployment, the bin sequence generated for a new input is fed into the trained HMM, which assigns an OOD score based on the learned transition probabilities and emission distributions. This enables OOD detection that leverages the shared structural characteristics of the network family while remaining computationally efficient. *Unlike conventional OOD detectors that provide a binary ID/OOD decision, our approach produces a quantitative measure of OOD likelihood, enabling the severity of distributional deviation to be assessed. More importantly, a single monitor can be shared across multiple networks within the same family, making OOD detection more scalable and efficient while reducing the overhead associated with deploying and maintaining OOD detection systems.*

Our technique is motivated by the works of Kornblith et al. (2019); Raghu et al. (2017), which investigate whether two networks learn similar representations by measuring the similarity between their raw activations on paired, equally sized input batches. Further, our approach is based on the hypothesis that feature representations at corresponding stages are similar across networks within the same family. *We empirically validate this hypothesis and find that the alignment between networks within the same family is* $5.25\times$ *higher than that observed across networks from different families.*

We further evaluate the generalizability of our approach using a leave-one-network-out (LONO) protocol. Specifically, when training the HMM, we randomly hold out one network and use it exclusively for testing, thereby evaluating whether the learned monitor can generalize to an unseen network of the same family. *Our results show that the monitor maintains high OOD detection performance on the held-out networks. Moreover, the performance of our HMM-based monitor on unseen networks is comparable to, and in some cases better than, that of the per-network monitors proposed in Lee et al. (2018); Henzinger et al. (2020), which construct a separate monitor for each network.*

Our main contributions are as follows:

1. We develop an end-to-end pipeline that computes per-stage anomaly scores, discretizes these scores into a fixed set of bins, represents each input as a sequence of bin indices, and trains an HMM as a family-level OOD monitor.
2. We empirically validate our hypothesis that networks within the same structural family exhibit greater similarity in their anomaly-score distributions than networks belonging to different families.
3. We empirically demonstrate the generalizability of our monitor using a leave-one-network-out (LONO) protocol, evaluating its performance on unseen networks.

Structure of the paper: Section 2 presents the related work. Section 3 introduces the terminology and describes our approach. Section 4 presents the experimental evaluation of our approach. Finally, Sections 5 and 6 discuss the limitations and conclusions of our work, respectively.

2 Related Work

Runtime monitoring has been explored for applications ranging from OOD detection to adversarial robustness monitoring Gupta et al. (2025). Since OOD detection is most closely related to our work, we focus our review on monitoring approaches for OOD detection.

Recent surveys Schotschneider et al. (2025); Torpmann-Hagen et al. (2025) provide comprehensive overviews of the field. Hendrycks & Gimpel (2017) use statistics of the softmax output to identify misclassified and OOD inputs. Kaur et al. (2023) propose a label-free approach for predicting the accuracy of DNN classifiers under distribution shifts without requiring labeled data from the target environment. Stoica et al. (2026) show that many existing OOD detection methods are vulnerable to gradient-based attacks. Their approach generates multiple perturbed versions of each input and uses the median softmax score as a reference for detecting potentially OOD inputs at runtime. In contrast, these approaches primarily rely on statistics of the network's outputs rather than its internal activation behavior.

Several approaches perform OOD detection using intermediate-layer representations rather than network outputs. Lee et al. (2018) propose a Mahalanobis distance-based confidence score by measuring the distance between an input's feature representation and the class-conditional feature distributions learned from training data. Cheng et al. (2019) record binary ReLU activation patterns produced by correctly classified training inputs and compare them with the activation patterns of new inputs at runtime to detect OOD samples. Henzinger et al. (2020) and Wu et al. (2024) construct box abstractions over feature-layer activations observed during training and reject inputs whose activations fall outside these boxes at runtime. Lukina et al. (2021) extend this idea by clustering feature representations and constructing a separate box abstraction for each cluster, thereby obtaining multiple boxes against which runtime activations are checked. Hashemi et al. (2021) model feature-layer activations using a multivariate Gaussian distribution and flag inputs whose activations deviate significantly from the learned distribution, this approach is further combined with outside-the-box monitoring in Hashemi et al. (2024). Boudardara et al. (2024) construct class-specific paths of active and inactive neurons at a feature layer from training data, identifying potential misclassifications when a new input follows a path associated with another class and OOD inputs when its path does not match any known path.

Our approach also leverages intermediate-layer activations, but differs from these methods in two key aspects. First, similar to Lee et al. (2018), we use a Mahalanobis-distance-based score, but instead of directly using the continuous score for detection, we discretize it into ordered severity

bins and model the resulting sequence of bin assignments across different stages of the network. Second, rather than constructing a separate monitor for each network, we build a single family-level monitor that can be applied across multiple networks belonging to the same architectural family. This enables the monitor to generalize across networks rather than being tied to a specific model.

Several works combine multiple monitoring mechanisms to detect different types of failures. Hacker & Seewig (2023) propose a runtime error-detection framework that combines complementary monitors to identify failures caused by ID, OOD, and adversarial inputs. Roy et al. (2022) introduce a two-level runtime monitoring architecture that combines a density model over a DNN's inputs, outputs, and explanation features with a graph Markov model capturing broader scene context to detect OOD and out-of-context inputs. Azeem et al. (2024) integrate multiple monitoring techniques into the Monitizer tool, providing a unified framework for runtime monitoring of DNNs.

3 METHOD

Our monitor built is in three stages: an outlier score is computed independently for each layer of a network (§3.1), the per-layer scores are reindexed along a depth axis that is comparable across networks of different depths and combined into a small number of *relative-depth stages* (§3.2), and a single hidden Markov model (HMM) Baum & Petrie (1966) is then fitted to the resulting stage sequences for an entire *structural family* of networks, rather than fitting a separate model for each network (§3.3).

We first build the monitor using the flag parameter `phase=BUILD`, and then use the resulting monitor to score new inputs with `phase=INFER`. Building the monitor involves computing and storing the parameters required for per-layer outlier scoring, relative-depth staging, and the family HMM. During inference, these stored parameters are used to compute the OOD score for a new input. The monitor is built once for each network family, while inference is performed independently for each new input, making the monitoring process efficient at deployment time. The key question is whether the pooled family HMM can generalize to network architectures that were not used during training, which we evaluate empirically in §4.2.

Terminology. A function $f : \mathbb{R}^m \to \mathbb{R}^n$ denotes a neural network that takes an input $\boldsymbol{x} \in \mathbb{R}^m$ and produces an output in $\mathbb{R}^n$. The network consists of L layers, where $\ell \in \{1, \ldots, L\}$ denotes the layer index, and $|\ell|$ denotes the number of neurons (or activation dimensions) at layer ℓ. Each layer implements a function α_ℓ, and for an input $\boldsymbol{x}$, the activation vector produced by layer ℓ is denoted as $\boldsymbol{a}^\ell_{\boldsymbol{x}}$, defined recursively as: $\boldsymbol{a}^\ell_{\boldsymbol{x}} = \alpha_\ell(\boldsymbol{a}^{\ell-1}_{\boldsymbol{x}}), \qquad \boldsymbol{a}^0_{\boldsymbol{x}} = \boldsymbol{x}$. Thus, the overall function f can be viewed as the composition of these layer functions: $f(\boldsymbol{x}) = \alpha_L(\alpha_{L-1}(\cdots \alpha_1(\boldsymbol{x}) \cdots))$. When the input is clear from context, we simply write $\boldsymbol{a}^\ell$ for the activation vector at layer ℓ. The in-distribution (ID) data used to train the monitor is denoted as $\mathcal{D}_{\text{ID}}$, and N denotes the number of ID inputs used for monitor training. For each ID input, the activation vector at layer ℓ is denoted as $\boldsymbol{a}^\ell_i \in \mathbb{R}^{|\ell|}$, where $i \in \{1, \ldots, N\}$. The activation matrix at layer ℓ, denoted as $\mathbf{A}^\ell \in \mathbb{R}^{N \times |\ell|}$, has its i-th row corresponding to the activation vector $\boldsymbol{a}^\ell_i$, $\mathbf{A}^\ell = \left[(\boldsymbol{a}^\ell_1), (\boldsymbol{a}^\ell_2), \ldots, (\boldsymbol{a}^\ell_N)\right]^\top$.

3.1 PER-LAYER OUTLIER SCORING

The goal of per-layer outlier scoring is to quantify how anomalous a new activation $\boldsymbol{a}^\ell$ is relative to the ID training activations at each layer. We first apply `KMeans` MacQueen (1967) to $\mathbf{A}^\ell$ for each layer ℓ, to cluster the ID activations into k clusters, where k is set to $\max(1, N/20)$ by default. Although we use `KMeans` in our experiments, the clustering step can be replaced by any suitable clustering algorithm. For each cluster, we fit a Ledoit–Wolf-shrunk Ledoit & Wolf (2004) covariance estimate $\hat{\boldsymbol{\Sigma}}_{\ell,c}$ using the centered training residuals of the activations assigned to that cluster. Shrinkage is useful because the number of samples in a cluster may be small relative to the dimensionality of the activations, which can result in ill-conditioned or singular covariance estimates. Using the estimated covariance $\hat{\boldsymbol{\Sigma}}_{\ell,c}$, we compute the Mahalanobis distance Mahalanobis (1936) of each training activation to its assigned cluster centroid. The Mahalanobis distance measures how far an activation is from the cluster centroid while accounting for the covariance structure of the cluster. Smaller distances indicate that an activation is closer to the centroid and is therefore less anomalous. We then compute $r_{\ell,c}$ as the p-th percentile of the Mahalanobis distances for each cluster, where $p = 95$ by default. At the end of the build phase, we store the cluster centroids, covariance

estimates, and percentile thresholds for each layer and cluster. During inference, we compute the Mahalanobis distance of a new activation $\boldsymbol{a}^\ell$ to each cluster centroid and select the nearest cluster. We normalize the resulting distance by the corresponding threshold $r_{\ell,c}$ to obtain the per-layer outlier score q^ℓ. A higher value of q^ℓ indicates that the activation is more anomalous relative to the ID training activations at that layer. All above steps are formalized in Algorithm 2 of Appendix A.1.

3.2 Relative-depth staging and the arity correction

Per-layer outlier scores computed in the previous section are not directly comparable across networks: layer index 6, for example, represents a very different position in a 6-layer network than in a 60-layer network. To make the scores comparable across networks of different depths, we divide the network into T stages. Each stage corresponds to a fraction of the network's depth. Each layer is assigned to a stage according to its relative depth, and the per-layer scores within each stage are aggregated to produce a single stage score. Each stage score is then discretized into one of M severity bins, resulting in a length-T sequence of bin indices for each input. Thus, each input is represented as a sequence of T discrete severity levels, which can be modeled using an HMM (§ 3.3).

This procedure takes the number of stages T, the number of severity bins M, and the quantile levels $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_{M-1})$ as hyperparameters. The quantile levels determine the bin boundaries for each stage based on the stage scores of the ID training samples. In our experiments, we set $T = 6$, $M = 6$, and $\boldsymbol{\rho} = (0.50, 0.70, 0.85, 0.95, 0.99)$, yielding six tail-dense severity bins. Given a network with L layers and a fixed number of stages T, shared across all networks and families, layer $\ell \in \{0, \ldots, L-1\}$ is assigned to a stage $t(\ell)$, according to $t(\ell) = \min(\lceil \frac{\ell}{L} \times T \rceil - 1, T - 1)$. Consequently, stage 0 always corresponds to the earliest relative-depth region of the network, while stage $T-1$ corresponds to the final region, regardless of the total number of layers L. For sufficiently shallow networks, one or more stages may contain no layers. We explicitly track such *empty stages* and exclude them from subsequent computations. To compute the score for each stage, we first z-score the layer-wise outlier scores independently for each layer. We then aggregate the normalized scores of the layers assigned to each stage by taking their *maximum*. We then provide an arity correction that accounts for the number of layers assigned to stage t, as described in detail below. Taking the maximum ensures that the stage score is determined by its most anomalous layer rather than being diluted by well-behaved layers within the same stage. Next, for each stage t, we compute the quantile boundaries $b_{t,1}, \ldots, b_{t,M-1}$ that define the M severity bins using the stage scores of the ID training samples. Each stage score is then assigned to one of these M severity bins according to the corresponding quantile boundaries. The final output is a length-T sequence of bin indices, $\mathbf{B} = (B_1, \ldots, B_T)$, for each input. All of the above steps are formalized in Algorithm 3 (RelativeDepth(n)) in Appendix A.2.

Arity correction. A maximum over k values is not directly comparable across stages with different values of k. For i.i.d. draws, the expected maximum increases with k as a purely statistical consequence of order statistics, independent of any actual difference in severity. Without correction, this systematically inflates the score of stages that contain more layers. Since different architectures may assign different numbers of layers to the same stage, this effect can confound the cross-architecture comparison that the staging scheme is intended to enable. We therefore correct for this effect by subtracting a Monte Carlo estimate (Metropolis & Ulam (1949)) of $\mathbb{E}[\max_{1 \le j \le k} Z_j]$, where $Z_j \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$ and k is the number of layers assigned to the stage, from the aggregated z-scored stage value.

3.3 Family-conditioned HMM

The central idea of our approach is to build *one monitor per structural family* of networks, rather than a separate monitor for each network. We define a structural family as a set of networks that share a common architectural motif (e.g., residual blocks or attention-based token mixing), while potentially differing in secondary architectural aspects such as width, activation function, normalization, or patch size. We obtain the stage sequences for all networks in a family by applying the per-layer outlier scoring and relative-depth staging procedures described in § 3.1 and 3.2. Each network contributes its own ID stage sequences to a pooled dataset, which is then used to fit a single HMM capturing the characteristic dynamics of the family. This enables the resulting monitor to generalize to new networks within the same structural family that were not used during training.

Algorithm 1 Family-Conditioned HMM

1: **Input:** Network family $\mathcal{F}$, phase $\in \{\text{BUILD}, \text{INFER}\}$, percentile threshold p
2: **if** phase = BUILD **then**
3: $\quad \mathcal{D}_{\text{train}} \leftarrow \emptyset$
4: $\quad$ **for** network $n \in \mathcal{F}$ **do**
5: $\quad\quad \mathbf{B}_n \leftarrow \text{RELATIVEDEPTH}(\, n \,)$ ▷ § 3.2
6: $\quad\quad \mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{\mathbf{B}_n\}$
7: $\quad$ **end for**
8: $\quad H_{\mathcal{F}} \leftarrow \text{TRAINHMM}(\mathcal{D}_{\text{train}})$ ▷ Baum–Welch EM, n_{init} random restarts
9: $\quad \mathcal{D}_{\text{val}} \leftarrow \emptyset$
10: $\quad$ **for** network $n \in \mathcal{F}$ **do**
11: $\quad\quad \mathbf{B}_n \leftarrow \text{RELATIVEDEPTH}(\, n \,)$ ▷ § 3.2
12: $\quad\quad \mathcal{D}_{\text{val}} \leftarrow \mathcal{D}_{\text{val}} \cup \{\mathbf{B}_n\}$
13: $\quad$ **end for**
14: $\quad \mathcal{L} \leftarrow \emptyset$
15: $\quad$ **for** sequence $\mathbf{B} \in \mathcal{D}_{\text{val}}$ **do**
16: $\quad\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{-\log P(\mathbf{B} \mid H_{\mathcal{F}})\}$
17: $\quad$ **end for**
18: $\quad \tau_{\text{ID}} \leftarrow \text{Percentile}_p(\mathcal{L}_{\text{train}})$ ▷ $p = 99$ in experiments
19: **else**
20: $\quad \mathbf{B} \leftarrow \text{RELATIVEDEPTH}(\, n \,)$ ▷ § 3.2
21: $\quad L \leftarrow -\log P(\mathbf{B} \mid H_{\mathcal{F}})$
22: $\quad$ **Output:** ID if $L < \tau_{\text{ID}}$, OOD otherwise
23: **end if**

We implement this procedure using Algorithm 1, which takes a family of networks $\mathcal{F}$, a phase (BUILD or INFER), and a percentile threshold p as input. In the build phase, we first collect the stage sequences from all networks in the family and fit an HMM $H_{\mathcal{F}}$ to the pooled sequences using the Baum–Welch expectation-maximization algorithm Baum et al. (1970) with multiple random restarts, as shown in Lines 3–8. We then compute the negative log-likelihood of each training sequence under the fitted HMM, as shown in line 16, and use these scores to determine the decision threshold τ_{ID} in line 18. Specifically, τ_{ID} is set to the p-th percentile of the negative log-likelihoods of the ID sequences, where p is a hyperparameter. This threshold is used to distinguish ID from OOD inputs during inference. We store the fitted HMM $H_{\mathcal{F}}$ and the threshold τ_{ID} for use in the inference phase. In the inference phase, we first compute the stage sequence for a new input using the previously stored parameters. We then calculate its negative log-likelihood under the family HMM $H_{\mathcal{F}}$. If the resulting score is below τ_{ID}, the input is classified as ID, otherwise, it is classified as OOD.

4 EXPERIMENTS

We describe the evaluation details of our monitor in this section. We consider three structural families of trained networks: Shallow CNNs LeCun et al. (1998), ResNet-style networks He et al. (2016), and Vision Transformers (ViTs) Dosovitskiy et al. (2021). We empirically investigate the following three research questions – **RQ1:** Do networks within the same structural family exhibit more closely aligned stage scores than networks from different families? **RQ2:** How does the OOD-detection performance of our tool compare with the baseline methods Lee et al. (2018) and Henzinger et al. (2020)? **RQ3:** Does our tool generalize to a network architecture that contributed no data during HMM fitting or threshold calibration? We perform all experiments on a 64-bit machine running Ubuntu 22.04 LTS with a 13th-generation Intel Core i5-1345U processor and 16 GiB of RAM.

4.1 BENCHMARKS AND EVALUATION PROTOCOL

Architectures and families. We trained 13 networks across three structural families: Shallow CNNs (5 networks), ResNet-style networks (5 networks), and Vision Transformers (ViTs) (3 networks). Networks within each family share the same defining structural design but differ in selected architectural properties, including depth, width, activation function, normalization scheme, internal block structure, and patch size. The Shallow CNNs were trained on MNIST LeCun et al. (1998) or Fashion-MNIST Xiao et al. (2017), while the ResNet-style networks and ViTs were trained on CIFAR-10 Krizhevsky (2009). Detailed architectural specifications are in Table 4 in Appendix B.

Datasets and OOD pairs. We classify the datasets used for evaluation into three categories: in-distribution (ID), near out-of-distribution (near-OOD), and far out-of-distribution (far-OOD). The distinction between near-OOD and far-OOD follows the OpenOOD benchmark (Wu et al., 2024) and is based on the degree of distribution shift relative to the ID data. For example, Wu et al. (2024) classifies FashionMNIST as near-OOD for MNIST because the two datasets share a similar visual domain, whereas CIFAR-10 is classified as far-OOD because it represents a substantially different visual domain. Consequently, near-OOD samples are generally more difficult to distinguish from ID samples, whereas far-OOD samples are easier to distinguish Wu et al. (2024). To evaluate the monitor under different degrees of distribution shift, we consider at least one near-OOD and one far-OOD dataset for each network family, as shown in Table 5 of Appendix B.

Metrics. We treat OOD samples as the positive cases and ID samples as the negative cases. We report three standard OOD-detection metrics: AUROC Fawcett (2006), the area under the receiver operating characteristic curve as the decision threshold varies; AUPR Davis & Goadrich (2006) the area under the precision–recall curve; and FPR@95%TPR Hendrycks & Gimpel (2017) the false-positive rate at the first operating point where the true-positive rate for OOD detection reaches at least 95%. Higher AUROC and AUPR values indicate better OOD-detection performance, whereas a lower FPR@95%TPR indicates better performance.

Monitor Building Parameters and Evaluation. We refer to the phase of computing stage sequences from raw activation values, described in §3.1 and §3.2, as *pipeline fitting*. The phase of fitting the family HMM and calibrating the threshold, described in §3.3, is referred to as *HMM fitting*. Finally, we refer to the evaluation of the monitor on new ID and OOD samples as *final evaluation*. We use mutually disjoint data splits for pipeline fitting, HMM fitting, threshold calibration, and final evaluation to reduce optimistic bias in the reported detection performance. For each network, we use $2,048$ ID samples for pipeline fitting, $2,048$ ID samples for HMM fitting, and 512 ID samples for threshold calibration. We use an additional 512 ID samples and 512 OOD samples for final evaluation. Pipeline fitting is performed independently for each network, whereas the sequences generated from the HMM-fitting and threshold-calibration splits are pooled across the contributing members of a family for HMM fitting and threshold calibration, respectively. The anomaly threshold is set to the 99th percentile of the pooled ID-validation anomaly scores, targeting an empirical false-positive rate of approximately 1% on the calibration data rather than optimizing detection accuracy. Lowering the threshold would detect more OOD inputs at the cost of more ID false alarms, whereas raising it would reduce false alarms but could miss more OOD inputs. Since the threshold is calibrated using only ID-validation scores, no OOD data are used for threshold selection.

4.2 Results

This section evaluates the monitors with respect to the three research questions outlined above. We empirically validate the hypothesis across networks within the same family, compare our approach against baseline methods, and assess its generalizability on unseen networks from the same family.

RQ1. Our approach is based on the hypothesis that networks within the same structural family exhibit more closely aligned stage scores than networks from different families. We empirically evaluate this hypothesis by comparing the stage scores of pairs of networks. For each pair of networks, we first compute the mean stage score over the ID validation samples for each stage populated in both networks (§ 3.2). We then compute the absolute difference between the corresponding mean stage scores and normalize it by the pooled standard deviation. We refer to this normalized measure as *diff/std*. Lower *diff/std* values indicate more closely aligned mean stage-score profiles, whereas higher values indicate greater separation. We report the minimum, mean, and maximum *diff/std* values across all valid stage comparisons for network pairs within and across families in Table 1.

The overall within-family mean diff/std is 0.135, compared with 0.710 across families, making the cross-family mean approximately $5.26\times$ larger. Moreover, every within-family group mean (0.109–0.141) is lower than every cross-family group mean (0.212–1.153). This pattern is not driven by a single family: even the largest within-family mean, 0.141 for ResNet, remains below the smallest cross-family mean, 0.212 for ResNet versus Shallow. These results support the hypothesis that networks within the same structural family exhibit more closely aligned mean stage-score profiles than networks belonging to different families. We also compare the within-family architectural variants to their respective base networks to identify which changes preserve alignment and which produce the greatest separation. Details in Appendix B.2.

Table 1: Stage-score alignment within and across network families measured by diff/std. Lower values indicate stronger alignment.

Network Family	**Min**	**Mean**	**Max**	n **Pairs**
Within-family				
Shallow	0.005	0.138	0.462	10
ResNet	0.003	0.141	0.594	10
ViT	0.003	0.109	0.314	3
Cross-family				
ResNet vs. Shallow	0.006	0.212	0.881	25
ResNet vs. ViT	0.475	1.004	1.826	15
Shallow vs. ViT	0.307	1.153	1.992	15
All within-family	—	**0.135**	**0.594**	—
All cross-family	—	**0.710**	**1.992**	—

Table 2: This table reports pooled-family AUROC for the HMM and mean per-network AUROC for the baselines. N and F denote near- and far-OOD sets, respectively.

Family	**OOD set**		**HMM**	**Maha.-single**	**Maha.-combined**	**Outside-the-Box**
Shallow	KMNIST Clanuwat et al. (2018)	N	0.990	0.964	0.991	0.742
Shallow	EMNIST Cohen et al. (2017)	N	0.989	0.946	0.987	0.642
Shallow	CIFAR-10 (gray)	F	0.988	0.895	0.998	0.981
Shallow	Noise	F	0.994	0.941	0.999	0.990
ResNet	SVHN Netzer et al. (2011)	F	0.924	0.840	0.948	0.924
ResNet	CIFAR-100	N	0.642	0.563	0.629	0.532
ViT	SVHN	F	0.846	0.886	0.916	0.801
ViT	CIFAR-100	N	0.600	0.641	0.623	0.528

RQ2: We compare family-conditioned HMM with the baseline methods Mahalanobis (Lee et al., 2018) and Outside-the-Box (Henzinger et al., 2020). All methods use the same data splits described in §4.1. Table 2 reports the pooled-family AUROC for the HMM and the mean AUROC for the baselines. The family-conditioned HMM outperforms single-layer Mahalanobis on all Shallow and ResNet evaluations, but not on the two ViT evaluations. Compared with Outside-the-Box, the HMM achieves a higher AUROC on seven of the eight network–OOD combinations.

All methods perform strongly on the four far-OOD evaluations, with AUROC values of at least 0.801. The largest improvements over Outside-the-Box occur on the near-OOD datasets, including Shallow/KMNIST (0.990 vs. 0.742), Shallow/EMNIST (0.989 vs. 0.642), and ResNet/CIFAR-100 (0.642 vs. 0.532). The near-OOD difficulty varies substantially across datasets. KMNIST and EMNIST remain comparatively easy for the Shallow family, with the HMM achieving AUROCs of 0.990 and 0.989, respectively. In contrast, CIFAR-100 is challenging for both CIFAR-10-trained families: all methods obtain AUROCs between 0.532 and 0.642 for ResNet and between 0.528 and 0.641 for ViT. Combined-layer Mahalanobis is generally the strongest baseline, outperforming family-conditioned HMM on six of the eight combinations. Nevertheless, family-conditioned HMM achieves the highest AUROC on ResNet/CIFAR-100 and slightly outperforms combined-layer Mahalanobis on Shallow/EMNIST. Overall, these results show that the family-pooled HMM is competitive with the per-network baselines, although it is not uniformly the strongest detector. Its principal advantage is the ability to use a single fitted HMM for an entire network family rather than requiring a separate fitted monitor for each network.

RQ3: Does the family HMM generalize to a network architecture that contributed no data during HMM fitting or threshold calibration? We evaluate this question using two experimental regimes for each family: *pooled-inclusive*, in which every family member contributes to HMM fitting and threshold calibration, and *leave-one-network-out (LONO)*, in which the evaluated network is excluded from both steps. The pooled-inclusive regime serves as the reference point for the LONO evaluation. In both regimes, one network is held out for evaluation, and the monitor is evaluated on that network. This procedure is repeated once for each network in the family, and the resulting per-network metrics are averaged to obtain the family-level results. Table 3 presents the results. The third and fourth columns report the mean AUROC for the pooled-inclusive and LONO regimes,

Table 3: Pooled-inclusive and LONO AUROC, averaged across member networks. The AUROC gap is the absolute difference between the two AUROC values. The final column reports the largest absolute difference between the two regimes across AUROC, AUPR, and FPR@95%TPR.

Family	OOD set	Pooled-incl. AUROC	LONO AUROC	AUROC gap	max |gap| (any metric)
Shallow	KMNIST	0.9886	0.9892	0.0006	0.0059 (FPR@95TPR)
Shallow	EMNIST	0.9946	0.9950	0.0004	0.0039 (FPR@95TPR)
Shallow	CIFAR-10 (gray)	0.9930	0.9958	0.0028	0.0057 (AUPR)
Shallow	Noise	0.9932	0.9970	0.0038	0.0098 (FPR@95TPR)
ResNet	SVHN	0.9389	0.9386	0.0003	0.0254 (FPR@95TPR)
ResNet	CIFAR-100	0.6533	0.6532	0.0001	0.1191 (FPR@95TPR)†
ViT	SVHN	0.8073	0.8168	0.0095	0.0410 (FPR@95TPR)
ViT	CIFAR-100	0.6103	0.6131	0.0028	0.0050 (AUPR)

respectively. The fifth column reports the absolute difference between the two regimes; a smaller difference indicates more consistent performance under LONO. Although the table reports the mean AUROC in these columns, we compute the corresponding differences for all three metrics: AUROC, AUPR, and FPR@95%TPR. The last column reports the maximum absolute difference among the three metrics and identifies the metric for which this maximum difference occurs.

The absolute AUROC difference is below 0.0095 for all combinations, ranging from 0.0001 to 0.0095. The AUROC of LONO regime is slightly higher than the pooled-inclusive AUROC in six of the eight combinations and slightly lower in the two ResNet combinations. These small changes, including occasional improvements under LONO, indicate that including the evaluated network in the family HMM fit provides no consistent AUROC advantage. ResNet/CIFAR-100 has a maximum gap of 0.1191 in FPR@95%TPR despite an AUROC difference of only 0.0001. Similarly, ViT/SVHN has an FPR@95%TPR gap of 0.0410, while its AUROC changes by only 0.0095. These cases show that operating-point metrics can vary more than threshold-independent ranking metrics and should therefore be interpreted separately from the near-identical AUROC results.

In summary, the LONO results show that a family HMM fitted and calibrated without any data from the evaluated network achieves nearly the same AUROC as the pooled-inclusive model. *This provides evidence that the family HMM generalizes to previously unseen networks within the same structural family. Thus, a single family-level monitor can be shared across multiple networks within the same family, avoiding the need to fit a separate HMM for each network.*

5 Limitations

Several components of our pipeline rely on fixed heuristic hyperparameters, including the numbers of clusters, stages, severity bins, and HMM hidden states, whose sensitivity has not been systematically evaluated. Relative-depth staging provides a common representation across networks, but layers at similar relative depths may not perform equivalent functions, and the Monte Carlo arity correction only approximates dependencies among layer scores. Moreover, the family-conditioned HMM does not uniformly outperform per-network baselines, particularly on challenging near-OOD datasets, as its primary advantage is monitor sharing rather than superior detection performance. As future work, we plan to evaluate larger collections of networks with independently controlled architectural variations to better characterize the boundaries of structural families. We also plan to extend the evaluation beyond image classification to other architectures, data modalities, tasks, and larger-scale deployment settings.

6 Conclusion

We presented a family-conditioned OOD monitoring pipeline for neural networks, in which a single HMM is fitted for each structural family rather than constructing a separate monitor for each network. We evaluated our approach across three structural families comprising thirteen networks and found that within-family alignment is substantially tighter than cross-family alignment. Our family-level monitor achieves OOD detection performance comparable to that of per-network baselines, while providing the additional benefit of generalizing across networks within the same family. We empirically validate this generalization using the LONO protocol, showing that performance on unseen networks remains close to that of the pooled-inclusive setting across all evaluated families and OOD datasets, including the challenging near-OOD cases.

AI USE STATEMENT

We used ChatGPT and GitHub Copilot for text polishing, including grammar correction and improving readability and flow. We also used Claude Code (Anthropic's agentic coding assistant built on Claude models) to design and implement the monitoring pipeline, develop experimental scripts, and assist with debugging, including identifying two silent bugs that had inverted early results before they were detected. All AI-assisted code was executed, and its outputs were carefully inspected and validated against the reported results. We take full responsibility for the final content of this work, including the text, claims, and artifacts produced with the assistance of generative AI tools.

ETHICS STATEMENT

This is a methods paper on out-of-distribution (OOD) monitoring for image classifiers, evaluated exclusively on standard public benchmark datasets, including MNIST, FashionMNIST, KMNIST, EMNIST, CIFAR-10, CIFAR-100, and SVHN, without the use of human-subject data. We do not identify any significant ethical concerns specific to this work beyond those generally associated with OOD-detection research, such as the risk that improved monitors could be used to justify unsafe deployments if their outputs are relied upon uncritically. We therefore do not believe that a more extensive ethics discussion is warranted.

REFERENCES

Muqsit Azeem, Marta Grobelna, Sudeep Kanav, Jan Křetínskỳ, Stefanie Mohr, and Sabine Rieder. Monitizer: automating design and evaluation of neural network monitors. In *International Conference on Computer Aided Verification*, pp. 265–279. Springer, 2024.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, and Mohamed Ghazel. Monitoring of neural network classifiers using neuron activation paths. In *International Conference on Verification and Evaluation of Computer and Communication Systems*, pp. 81–96. Springer, 2024.

Chih-Hong Cheng, Georg Nührenberg, and Hirotoshi Yasuoka. Runtime monitoring neuron activation patterns. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 300–303. IEEE, 2019.

Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240, 2006.

Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, et al. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature medicine*, 24(9):1342–1350, 2018.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pp. 1321–1330. PMLR, 2017.

Ashutosh Gupta, Thomas A Henzinger, Konstantin Kueffner, Kaushik Mallik, and David Pape. Monitoring robustness and individual fairness. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 790–801, 2025.

Lukas Hacker and Jörg Seewig. Insufficiency-driven dnn error detection in the context of sotif on traffic sign recognition use case. *IEEE Open Journal of Intelligent Transportation Systems*, 4: 58–70, 2023.

Vahid Hashemi, Jan Křetínskỳ, Stefanie Mohr, and Emmanouil Seferis. Gaussian-based runtime detection of out-of-distribution inputs for neural networks. In *International Conference on Runtime Verification*, pp. 254–264. Springer, 2021.

Vahid Hashemi, Jan Křetínskỳ, Sabine Rieder, Torsten Schön, and Jan Vorhoff. Gaussian-based and outside-the-box runtime monitoring join forces. In *International Conference on Runtime Verification*, pp. 218–228. Springer, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Hkg4TI9xl.

Thomas A. Henzinger, Anna Lukina, and Christian Schilling. Outside the box: Abstraction-based monitoring of neural networks. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 2433–2440. IOS Press, 2020. doi: 10.3233/FAIA200375. URL https://doi.org/10.3233/FAIA200375.

Heinrich Jiang, Been Kim, Melody Guan, and Maya Gupta. To trust or not to trust a classifier. *Advances in neural information processing systems*, 31, 2018.

Ramneet Kaur, Susmit Jha, Anirban Roy, Oleg Sokolsky, and Insup Lee. Predicting out-of-distribution performance of deep neural networks using model conformance. In *2023 IEEE International Conference on Assured Autonomy (ICAA)*, pp. 19–28. IEEE, 2023.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529. PMLR, 2019. URL https://proceedings.mlr.press/v97/kornblith19a.html.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of multivariate analysis*, 88(2):365–411, 2004.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pp. 7167–7177, Montréal, Canada, 2018.

Anna Lukina, Christian Schilling, and Thomas A Henzinger. Into the unknown: Active monitoring of neural networks. In *International Conference on Runtime Verification*, pp. 42–61. Springer, 2021.

James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pp. 281–297. University of California Press, 1967.

Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936.

Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 4. Granada, 2011.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 6076–6085. Curran Associates, Inc., 2017.

Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. *Advances in neural information processing systems*, 32, 2019.

Anirban Roy, Adam Cobb, Nathaniel Bastian, Brian Jalaian, and Susmit Jha. Runtime monitoring of deep neural networks using top-down context models inspired by predictive processing and dual process theory. In *AAAI Spring Symposium 2022*, 2022.

Albert Schotschneider, Svetlana Pavlitska, and J Marius Zöllner. Runtime safety monitoring of deep neural networks for perception: A survey. In *2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 6116–6121. IEEE, 2025.

Maria Stoica, Abdelrahman Hekal, and Alessio Lomuscio. A robust out-of-distribution detection framework via synergistic smoothing. *arXiv preprint arXiv:2605.08191*, 2026.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Birk Torpmann-Hagen, Michael A Riegler, Pål Halvorsen, and Dag Johansen. Runtime verification for deep learning systems. In *ENASE*, pp. 367–377, 2025.

Changshun Wu, Weicheng He, Chih-Hong Cheng, Xiaowei Huang, and Saddek Bensalem. BAM: Box abstraction monitors for real-time ood detection in object detection. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2632–2638. IEEE, 2024. doi: 10.1109/IROS58592.2024.10801584.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

APPENDIX

A METHOD

This appendix provides the complete algorithmic specification of the monitoring pipeline introduced in Section 3. Algorithm 1 describes the family-conditioned HMM procedure, while Algorithms 2 and 3 specify the per-layer outlier-scoring and relative-depth sequence-construction procedures, respectively. For each component, we distinguish between the build phase, in which the required parameters are estimated and stored using ID data, and the inference phase, in which these parameters are applied to a new input. The algorithms presented here complement the conceptual description in the main text by specifying the inputs, outputs, stored parameters, and computational steps of the complete pipeline.

A.1 PER-LAYER OUTLIER SCORING

Algorithm 2 provides the complete build and inference procedures for the per-layer outlier-scoring component introduced in Section 3.1. During the build phase, the algorithm takes as input the ID activation matrices $\{A^\ell\}_{\ell=1}^{L}$, the number of clusters k, and the percentile p. For each monitored layer ℓ, KMeans is applied to the corresponding activation matrix A^ℓ to obtain the cluster centroids $\{\mu_{\ell,c}\}_{c=1}^{k}$ and the assignment of each training activation to a cluster (line 5).

For every cluster c, the index set $I_{\ell,c}$ identifies the training activations assigned to that cluster (line 5). A Ledoit–Wolf-shrunk covariance estimate $\hat{\Sigma}_{\ell,c}$ is then fitted to the centered residuals of these activations (line 8). The Mahalanobis distance from each assigned training activation to the cluster

Algorithm 2 Per-layer outlier scoring

1: **Input:** Phase $\in \{\text{BUILD}, \text{INFER}\}$
2: **if** Phase = BUILD **then**
3: **Input:** ID activation matrices $\{\mathbf{A}^\ell\}_{\ell=1}^{L}$, where $\mathbf{A}^\ell \in \mathbb{R}^{N\times|\ell|}$; number of clusters k; percentile p
4: **for** $\ell = 1$ to L **do**
5: $\{\boldsymbol{\mu}_{\ell,c}\}_{c=1}^{k}, \{c_i^\ell\}_{i=1}^{N} \leftarrow \text{KMeans}(\mathbf{A}^\ell, k)$ ▷ cluster centroids and assignments
6: **for** $c = 1$ to k **do**
7: $\mathcal{I}_{\ell,c} \leftarrow \{i \in \{1,\dots,N\} : c_i^\ell = c\}$
8: $\hat{\boldsymbol{\Sigma}}_{\ell,c} \leftarrow \text{LW-Cov}\left(\{\boldsymbol{a}_i^\ell - \boldsymbol{\mu}_{\ell,c} : i \in \mathcal{I}_{\ell,c}\}\right)$ ▷ Ledoit–Wolf-shrunk covariance
9: **for** $i \in \mathcal{I}_{\ell,c}$ **do**
10: $d_i^{\ell,c} \leftarrow \sqrt{(\boldsymbol{a}_i^\ell - \boldsymbol{\mu}_{\ell,c})^\top \hat{\boldsymbol{\Sigma}}_{\ell,c}^{-1} (\boldsymbol{a}_i^\ell - \boldsymbol{\mu}_{\ell,c})}$ ▷ Mahalanobis Distance
11: **end for**
12: $r_{\ell,c} \leftarrow \text{Percentile}_p\left(\{d_i^{\ell,c} : i \in \mathcal{I}_{\ell,c}\}\right)$
13: **end for**
14: **end for**
15: **Output:** $\mathcal{P}_A = \{\boldsymbol{\mu}_{\ell,c}, \hat{\boldsymbol{\Sigma}}_{\ell,c}, r_{\ell,c}\}_{\ell=1,c=1}^{L,k}$
16: **else**
17: **Input:** New input activation $\boldsymbol{a}^\ell \in \mathbb{R}^{|\ell|}$ of an input at each layer ℓ; stored parameters $\mathcal{P}_A$
18: **for** $\ell = 1$ to L **do**
19: **for** $c = 1$ to k **do**
20: $d_{\ell,c}(\boldsymbol{a}^\ell) \leftarrow \sqrt{(\boldsymbol{a}^\ell - \boldsymbol{\mu}_{\ell,c})^\top \hat{\boldsymbol{\Sigma}}_{\ell,c}^{-1} (\boldsymbol{a}^\ell - \boldsymbol{\mu}_{\ell,c})}$
21: **end for**
22: $c_\ell^* \leftarrow \arg\min_{c\in\{1,\dots,k\}} d_{\ell,c}(\boldsymbol{a}^\ell)$
23: $q^\ell \leftarrow \dfrac{d_{\ell,c_\ell^*}(\boldsymbol{a}^\ell)}{r_{\ell,c_\ell^*}}$
24: **end for**
25: **Output:** $\{q^\ell\}_{\ell=1}^{L}$
26: **end if**

centroid is computed in line 10. The cluster-specific normalization threshold $r_{\ell,c}$ is set to the p-th percentile of these distances (line 12). The build phase stores the centroids, covariance estimates, and percentile thresholds for all monitored layer–cluster pairs as

$$P_A = \{\mu_{\ell,c}, \hat{\Sigma}_{\ell,c}, r_{\ell,c}\}_{\ell=1,c=1}^{L,k}.$$

A.2 Relative-Depth Staging and the Arity Correction

Algorithm 3 specifies the build and inference procedures for constructing the relative-depth sequences introduced in Section 3.2. Its hyperparameters are the number of relative-depth stages T, the number of severity bins M, and the quantile levels

$$\rho = (\rho_1, \ldots, \rho_{M-1}).$$

In our experiments, we use $T = 6$, $M = 6$, and $\rho = (0.50, 0.70, 0.85, 0.95, 0.99)$.

During the build phase, the mean μ_{q_ℓ} and standard deviation σ_{q_ℓ} of the ID outlier scores are computed independently for each monitored layer ℓ (line 5). Each layer is then assigned to a relative-

Algorithm 3 Relative-Depth Sequence Construction RELATIVEDEPTH(n)

Input: Phase $\in \{\text{BUILD}, \text{INFER}\}$, number of stages T, number of severity bins M, quantile levels $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_{M-1})$
if phase = BUILD **then**
  Input: Layer-wise scores $\{\{q_i^\ell\}_{i=1}^N\}_{\ell=1}^L$
  for $\ell = 1$ to L **do**
    $\mu_\ell^q, \sigma_\ell^q \leftarrow \text{mean}(\{q_i^\ell\}_{i=1}^N), \text{std}(\{q_i^\ell\}_{i=1}^N)$
    $t(\ell) \leftarrow \min\big(\lceil \frac{\ell}{L} T \rceil - 1, T - 1\big)$
  end for
  for $t = 0$ to $T - 1$ **do**
    $\mathcal{L}_t \leftarrow \{\ell : t(\ell) = t\}$
    if $\mathcal{L}_t \neq \emptyset$ **then**
      for $i = 1$ to N **do**
        $s_{i,t} \leftarrow \max_{\ell \in \mathcal{L}_t} \dfrac{q_i^\ell - \mu_\ell^q}{\sigma_\ell^q} - C(|\mathcal{L}_t|)$ ▷ Substracting Monte-Carlo Estimate
      end for
      $(b_{t,1}, \ldots, b_{t,M-1}) \leftarrow \text{Quantiles}_{\boldsymbol{\rho}}(\{s_{i,t}\}_{i=1}^N)$
      for $i = 1$ to N **do**
        $B_{i,t} \leftarrow \text{Bin}(s_{i,t}; b_{t,1}, \ldots, b_{t,M-1})$
      end for
    end if
  end for
  Output: $\{\mu_\ell^q, \sigma_\ell^q\}_{\ell=1}^L, \{t(\ell)\}_{\ell=1}^L, \{b_{t,j}\}_{t,j}, \{\mathbf{B}_i\}_{i=1}^N$
else
  Input: Layer-wise scores $\{q^\ell\}_{\ell=1}^L$ of an input
  for $\ell = 1$ to L **do**
    $z^\ell \leftarrow \dfrac{q^\ell - \mu_\ell^q}{\sigma_\ell^q}$
  end for
  for $t = 0$ to $T - 1$ **do**
    $\mathcal{L}_t \leftarrow \{\ell : t(\ell) = t\}$
    if $\mathcal{L}_t \neq \emptyset$ **then**
      $s_t \leftarrow \max_{\ell \in \mathcal{L}_t} z^\ell - C(|\mathcal{L}_t|)$ ▷ Substracting Monte-Carlo Estimate
      $B_t \leftarrow \text{Bin}(s_t; b_{t,1}, \ldots, b_{t,M-1})$
    end if
  end for
  Output: $\mathbf{B} = (B_1, \ldots, B_T)$
end if

depth stage according to

$$t(\ell) = \min\left(\left\lceil \frac{\ell}{L} \times T \right\rceil - 1, T - 1\right),$$

as specified in line 6. The set of layers assigned to stage t is denoted by

$$L_t = \{\ell : t(\ell) = t\}.$$

Stages for which $L_t = \emptyset$ are explicitly identified and excluded from subsequent computations.

For each populated stage, the layer-wise scores are standardized using the stored layer-specific statistics. The stage score for ID sample i is then computed as

$$s_{i,t} = \max_{\ell \in L_t} \left(\frac{q_i^\ell - \mu_{q_\ell}}{\sigma_{q_\ell}} \right) - C(|L_t|),$$

where $C(|L_t|)$ is the arity-correction term defined below. The quantile boundaries $b_{t,1}, \ldots, b_{t,M-1}$ are estimated from the resulting ID stage scores (line 14), after which each score is assigned to one of the M severity bins (line 16). The build phase stores the layer-wise normalization statistics, the layer-to-stage assignments, and the stage-specific bin boundaries.

During inference, the new input's layer-wise outlier scores are standardized using the stored values μ_{q_ℓ} and σ_{q_ℓ} (line 24). The corrected maximum is then computed for every populated stage, and the resulting stage score is discretized using the corresponding stored boundaries (lines 26–32). This produces the discrete stage sequence

$$B = (B_1, \ldots, B_T),$$

with empty stages omitted from the computations as specified in the algorithm. The resulting sequence is used as the observation sequence for the family-conditioned HMM.

Arity correction. The correction term used in lines 12 and 29 is

$$C(k) = \widehat{\mathbb{E}}\left[\max_{1 \le j \le k} Z_j\right], \qquad Z_j \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1),$$

where $k = |L_t|$ is the number of layers assigned to stage t. We estimate $C(k)$ using Monte Carlo sampling and subtract it from the maximum standardized layer score. The value of the correction therefore depends only on the number of layers assigned to the stage. For a stage containing one layer, $C(1)$ is approximately zero; for stages containing multiple layers, the correction compensates for the increase in the expected maximum induced by the larger number of terms. The same correction values are used during the build and inference phases.

B Experiments

This appendix provides supplementary details on the experimental setup and analyses presented in Section 4. Appendix B.1 specifies the architectures, monitored-layer conventions, training datasets, and OOD test sets used in the evaluation. Appendix B.2 provides the detailed base-to-variant alignment analysis that complements the aggregate within-family and cross-family results reported in Section 4.2. The experimental protocols, data-split definitions, evaluation metrics, and principal detection results remain in the main text.

B.1 Architectures and Families

Table 4 provides the detailed specifications of the 13 networks used in the experiments. The networks are organized into three structural families: five Shallow CNNs, five ResNet-style networks, and three Vision Transformers. For each network, the table reports the number of monitored layers L, held-out ID test accuracy, training dataset, and architectural properties that differ from the designated base network of the family.

The architectural variants are designed to introduce controlled differences within each family. Some variants modify one property, such as activation function, normalization scheme, patch size, or depth. Other variants modify multiple properties simultaneously. The "Axis varied vs. family base"

Table 4: Architectures of the neural networks used in this study. L denotes the number of layers.

Family	Network	L	Test acc.	Dataset	Axis varied vs. family base
Shallow	Shallow-Base	4	90.4%	FashionMNIST	— (base)
Shallow	Shallow-Base	4	98.7%	MNIST	dataset only
Shallow	Shallow-Depth3	3	91.2%	FashionMNIST	depth, width, kernel size
Shallow	Shallow-GELU+Drop	4	98.7%	MNIST	activation (ReLU→GELU) + dropout
Shallow	Shallow-GELU	4	98.8%	MNIST	activation (ReLU→GELU) only
ResNet	ResNet-Base	13	85.4%	CIFAR-10	— (base)
ResNet	ResNet-Width	9	82.6%	CIFAR-10	block count + width
ResNet	ResNet-Bottleneck	7	66.5%	CIFAR-10	per-block microarchitecture (bottleneck)
ResNet	ResNet-GroupNorm	13	82.5%	CIFAR-10	normalization (BatchNorm→GroupNorm)
ResNet	ResNet-GELU	13	85.9%	CIFAR-10	activation (ReLU→GELU)
ViT	ViT-Base	6	80.2%	CIFAR-10	— (base)
ViT	ViT-Patch8	6	75.6%	CIFAR-10	patch size ($4 \times 4 \rightarrow 8 \times 8$)
ViT	ViT-Depth8	8	80.6%	CIFAR-10	depth (6→8 blocks)

Table 5: ID datasets and OOD test pairs, per family.

Family	ID dataset(s)	OOD test set	Near / far
Shallow	MNIST, FashionMNIST	KMNIST	Near
Shallow	MNIST, FashionMNIST	EMNIST (letters split)	Near
Shallow	MNIST, FashionMNIST	CIFAR-10 (grayscale, resized)	Far
Shallow	MNIST, FashionMNIST	Uniform noise	Far
ResNet	CIFAR-10	CIFAR-100 Krizhevsky (2009)	Near
ResNet	CIFAR-10	SVHN	Far
ViT	CIFAR-10	CIFAR-100	Near
ViT	CIFAR-10	SVHN	Far

column in Table 4 explicitly identifies these cases and should therefore be used when interpreting the corresponding base-to-variant alignment results in Table 6.

For example, Shallow-Base and Shallow-Depth3 differ jointly in depth, width, and convolutional kernel size. Shallow-Base contains four monitored layers, including a layer with $1,568$ activation dimensions, whereas Shallow-Depth3 contains three monitored layers, including a corresponding layer with $2,352$ activation dimensions. Their comparison therefore represents a combined architectural modification and does not isolate the effect of depth alone. The same interpretation applies to ResNet-Width, which changes both block count and width relative to ResNet-Base.

Table 5 lists the ID datasets and OOD test sets associated with each family. The Shallow family contains networks trained on MNIST or FashionMNIST and is evaluated using KMNIST and EMNIST as near-OOD sets and grayscale CIFAR-10 and uniform noise as far-OOD sets. The ResNet and ViT families are trained on CIFAR-10 and evaluated using CIFAR-100 as near-OOD and SVHN as far-OOD.

Monitored-layer convention. The value L reported in Table 4 follows a family-specific but internally consistent hook-point convention. For the Shallow and ResNet families, a layer contributes to L when a convolutional or linear operation is immediately followed by a nonlinearity. Normalization, pooling, and residual-addition operations are treated as transparent and are not counted as separate monitored layers. For ViTs, monitoring is performed at transformer-block granularity, so L is equal to the number of transformer blocks. This convention determines the activation sequence supplied to Algorithm 2 and the layer-to-stage assignments constructed by Algorithm 3.

Family-level structural distinctions. The family assignments reflect the primary computational motif of each architecture. Shallow CNNs use convolutional processing without skip connections. ResNet-style networks organize convolutional operations within residual blocks. ViTs perform token mixing using self-attention and contain residual connections around their attention and MLP sublayers. Thus, the ViT family differs from the ResNet family primarily in its token-mixing mechanism rather than in the presence or absence of residual connections. The alignment analyses in Section 4.2 and Appendix B.2 evaluate whether these family-level similarities and differences are reflected in the activation-derived stage scores.

B.2 Supplementary Alignment Results

This section provides supplementary analyses for RQ1, complementing the aggregate alignment results reported in Section 4.2. We first clarify the number of within-family and cross-family network pairs contributing to Table 1. We then examine the alignment between each family's base network and selected architectural variants to identify which of the evaluated modifications preserve or weaken the stage-score alignment. The principal OOD-detection and leave-one-network-out results for RQ2 and RQ3 are reported in Section 4.2.

Pairwise comparison counts. Table 1 aggregates pairwise alignment results across all 13 networks. For a family containing m networks, the number of unique within-family pairs is $\binom{m}{2}$. The five Shallow networks and five ResNet-style networks therefore contribute 10 within-family pairs each, while the three ViTs contribute three within-family pairs. For cross-family comparisons, every network in one family is paired with every network in the other family. This gives 25 ResNet–Shallow pairs, 15 ResNet–ViT pairs, and 15 Shallow–ViT pairs.

The minimum, mean, and maximum values in Table 1 are computed over the valid stage-level diff/std values for these network pairs. Only stages populated in both members of a pair are included. The aggregate values and their interpretation are discussed in Section 4.2. Here, we focus on the variation among selected base-to-variant comparisons, which is not visible from the family-level aggregates alone.

Base-to-variant alignment. Table 6 reports the pairwise diff/std between each family's designated base network and selected architectural variants. Lower values indicate closer alignment of their mean stage-score profiles. Because some variants modify more than one architectural property, the first column identifies the complete set of properties changed relative to the corresponding base network.

Among the evaluated variants, the smallest diff/std is observed when replacing ReLU with GELU in the ResNet family, with a value of 0.021. Replacing BatchNorm with GroupNorm in ResNet gives 0.046, while replacing ReLU with GELU in the Shallow family gives 0.055. Changing the ViT patch size from 4×4 to 8×8 gives 0.062. These variants therefore remain comparatively close to their respective base networks under the stage-score alignment measure.

The variants involving changes to depth, width, or internal block structure produce larger differences. Shallow-Depth3, which jointly modifies depth, width, and kernel size, gives a diff/std of 0.063. Increasing the ViT depth from six to eight blocks gives 0.115, and jointly changing the ResNet block count and width gives 0.153. The largest difference among the comparisons in Table 6 is 0.230 for the ResNet bottleneck variant, which changes the per-block microarchitecture.

Within the set of architectures evaluated here, these results indicate that the tested activation-function, normalization, and patch-size changes preserve closer alignment with the corresponding base networks than the tested changes to depth, width, and per-block microarchitecture. However, the comparisons cover only a small number of variants, and several rows combine multiple architectural changes. The results should therefore be interpreted as observations about the evaluated networks rather than as general effects of these architectural properties.

Table 6: **Axis-isolation analysis**: pairwise diff/std between each family's base network and selected architectural variants, sorted from most to least alignment-preserving. Some variants modify more than one architectural property, as indicated in the first column.

Axis varied	**Family (base vs. variant)**	**diff/std**
Activation (ReLU→GELU)	ResNet (Base vs. ResNet-GELU)	0.021
Normalization (BatchNorm→GroupNorm)	ResNet (Base vs. ResNet-GroupNorm)	0.046
Activation (ReLU→GELU)	Shallow (Base vs. Shallow-GELU)	0.055
Patch size ($4 \times 4 \rightarrow 8 \times 8$)	ViT (Base vs. ViT-Patch8)	0.062
Depth, width, kernel size	Shallow (Base vs. Shallow-Depth3)	0.063
Depth (6→8 blocks)	ViT (Base vs. ViT-Depth8)	0.115
Block count + width	ResNet (Base vs. ResNet-Width)	0.153
Per-block microarchitecture (bottleneck)	ResNet (Base vs. ResNet-Bottleneck)	0.230