Hierarchical Sparse Neural Networks for Structure-Aware Ransomware Detection Under Distribution Shift

Isaac Kofi Nti *

Posted Date: 17 April 2026

doi: 10.20944/preprints202604.1252.v1

Keywords: behavioral ransomware detection; hierarchical sparse neural networks; open-set recognition; model calibration; distribution shift robustness; feature interpretability; MLRan dataset

 doi:10.20944/preprints202604.1252.v1

Article

Hierarchical Sparse Neural Networks for Structure-Aware Ransomware Detection Under Distribution Shift

Isaac Kofi Nti [1,2]

[1] School of Information Technology, University of Cincinnati, Cincinnati, Ohio, USA; ntious1@gamil.com

[2] Information Technology and Analytics Center (ITAC), School of Information Technology, University of Cincinnati, Cincinnati, Ohio, USA

Abstract

Behavioral ransomware detection often achieves high accuracy in standard evaluations; however, these results frequently fail to generalize under distribution shifts or when encountering previously unseen families. This study evaluates detection performance on the MLRan dataset (4,880 samples across 64 families) using four rigorous evaluation protocols: stratified, temporal, family-disjoint, and open-set. To ensure a strict separation of learned features, the family-disjoint and open-set splits were executed at the family level. We propose the Hierarchical Sparse Neural Network (HSNN), a taxonomy-aligned model with group-level and branch-level gating for structured interpretability. Unlike flat architecture, HSNN introduces a hierarchical gating mechanism aligned with a predefined behavioral taxonomy, enabling structured interpretability and modality-level analysis. The baseline FlatMLP had a slightly higher average macro-F1 score (0.9860 vs. HSNN's 0.9839), but the HSNN was better calibrated and more parameter efficient. The HSNN reduced calibration error by 34.1% (absolute reduction of 0.0056 in ECE) and model complexity by 42% in terms of parameter count. HSNN showed slightly lower variability than FlatMLP and broadly stable gate patterns across seeds. The proposed HSNN achieved one of the highest performances under the paper's open-set family protocol (0.9930 vs. 0.9913) using a maximum-softmax novelty baseline. Our feature analysis shows that string-based artifacts remain strong predictors, but the HSNN's hierarchical structure encourages a more balanced weighting across behavioral modalities, reducing reliance on any single feature type. These results indicate that structured, sparse architecture presents a competitive and well-calibrated alternative to conventional dense models under the evaluated settings.

Keywords: behavioral ransomware detection; hierarchical sparse neural networks; open-set recognition; model calibration; distribution shift robustness; feature interpretability; MLRan dataset

1. Introduction

Ransomware is considered one of the most dangerous types of malware [1–6]. Unlike other types of malware which typically carry out actions such as stealing information or performing other malicious operations, the primary purpose of ransomware is to gain illicit profit from victims [1,3,7]. The way ransomware works is by encrypting victims' information with cryptographic algorithms and threatening to publish users' personal photos, videos or sensitive business information unless they pay a ransom to get it decrypted. Whatever the choice of victims, in most cases, ransomware imposes some costs to its victims whether they pay the ransom or not [6,8,9]. As stated in the 2025 Verizon Data Breach Investigations Report by Verizon, 44% of the breaches they investigated had ransomware which is a 12% increase from the 32% reported in 2024 [10]. Thus, ransomware remains a rapidly evolving and significant cyber threat, accounting for a substantial proportion of reported data breaches globally.

 doi:10.20944/preprints202604.1252.v1

According to the FBI's Internet Crime Complaint Center (IC3), ransomware remains one of the most financially damaging cyber threats, with billions of dollars in reported losses annually [7]. For example, the global average cost of a data breach is estimated at approximately $4.88 million, according to the IBM cost of a data breach report. In the specific context of ransomware, recovery costs alone average around $1.5 million, excluding ransom payments, as reported by Sophos [10–12]. Critical sectors that have been targeted including the healthcare, energy and financial services sectors have been subjected to attacks that are so severe they even prompt national policy responses [1–6,8,10–13]. All these make it essential for better and more advanced intrusion detection systems (IDSs) that can, for example, help in the early detection of ransomware activities in the different stages of an attack [14].

In addition to these trends, the technical nature of the threat has changed substantially in the last ten years. The first families of ransomware that came out in the 2010s, like CryptoLocker, and CryptoWall, were simple in their operation. They encrypted files with built-in symmetric keys and asked for payment through centralized channels [5,15,16]. But the rise of Ransomware-as-a-Service (RaaS) platforms has made it much easier for even low-skilled hackers to start ransomware attacks using pre-made toolkits and partner-based infrastructures [2,17,18]. New technologies, like artificial intelligence, could make it even easier to automate attacks, but the main reason this is happening is that ransomware resources are becoming more common.

Because of this, attackers can go after a lot of people and businesses at once, and their ransom demands can be in the millions of dollars [5,19]. Also, top groups like LockBit, BlackCat/ALPHV, and RansomHub have turned this model into very organized cybercriminal ecosystems by using affiliate networks and planned blackmail strategies [17,20,21]. LockBit ransomware, for instance, employs advanced evasion techniques such as API hashing and string obfuscation to hinder static and dynamic analysis [20,21]. In addition, it has been reported to facilitate lateral movement through mechanisms such as the use of stolen credentials and propagation across network shares [20,21].

As a result, these changes make it much harder for detection systems that use static signatures for detection [22]. These kinds of methods are reactive by nature, relying on patterns from samples that have already been analyzed. They can also be easily avoided by obfuscation, packing, or small code changes. Consequently, content-based methods find it hard to find new or zero-day malware. Because of this limit, researchers have turned their attention to behavior-based detection, which looks at how malware works instead of how it looks [9,23–25]. In this context, the execution of sample malware in a controlled environment is dynamically monitored to record its interactions with the underlying system, encompassing API calls, registry modifications, file operations, network activity, and process creation [23]. These behavioral traces create a more stable fingerprint because the main bad actions stay the same even when the code structure changes. So, API sequence-based features are very strong against common ways to get around them [26]. However, the number of ransomware cases is still rising quickly around the world.

Fortunately, the Machine Learning (ML) community has addressed this challenge with multiple studies on ML techniques to detect ransomware [27,28]. We used traditional models like Logistic Regression, Random Forests, and Gradient-Boosted Trees, along with deep learning methods [5], to analyze behavioral features found in sandboxed execution traces. For instance, API call sequences have been used to model how processes work. Long Short-Term Memory (LSTM) networks are built to find patterns over time, which helps lessen the impacts of noise and delays [1,5]. Recent studies have progressed this field by employing convolutional, graph-based, and transformer-based architectures, yielding promising outcomes [3,4,6,23].

Even with these improvements, literature [5,22,29–32] points out problems that keep coming up that make people worry about how useful the reported findings are and how well the proposed systems will work in the real world.

First, publicly available datasets for ransomware detection are still small, not very diverse, and don't cover a long enough period of time. Most of them only have a few hundred samples [31,33].

 doi:10.20944/preprints202604.1252.v1

This makes it hard to reproduce results and compare studies, since many of the results depend on private or poorly documented data and evaluation setups.

Second, behavioral features are often treated as flat vectors, which means that signals like API calls, changes to the registry, and file activity are combined without keeping track of how they are related. This could cause a loss of context and make generalization less strong.

Third, existing ransomware detection models achieve high performance but provide limited insight into how different behavioural modalities contribute to detection decisions, especially under distribution shifts such as unseen families. Thus, important practical issues like calibration and stability across random seeds are still not well understood, even though they have a direct impact on reliability and interpretability in real-world situations.

The new MLRan dataset [33] helps with the lack of data by giving us a bigger and more varied benchmark with more than 4,800 samples, 64 ransomware families, and balanced ransomware and goodware classes. This dataset also has a lot of behavioral features from dynamic analysis. Prior research [33] indicates strong performance with classical machine learning models; however, it fails to evaluate calibration, robustness, or sensitivity to experimental conditions, nor does it investigate the potential for more structured behavioral representations to provide further insights.

Thus, in this study, we seek to address the above-mentioned gaps. We develop a Hierarchical Sparse Neural Network (HSNN) model for behavioral ransomware detection using multi-modal features with known structure. Rather than treating behavioral features as a flat vector, the proposed HSNN organizes them in a two-level hierarchy aligned with the MLRan taxonomy. The lower-level encoders capture fine-grained signals, while higher-level encoders bring these signals together into broader behavioral patterns. The model uses learned gates with sparsity to focus on the most informative feature groups, while reducing overfitting, especially in the STRING features. Additionally, HSNN gate values make it possible to see which features the model relies on, and how stable those choices are across different runs. We evaluate HSNN against a range of classical and neural baselines, including logistic regression, ElasticNet, random forests, XGBoost, and MLP variants, under several protocol settings such as stratified splits, time-aware evaluation, family-disjoint testing, and open-set scenarios. In addition to standard performance metrics, we examine calibration (expected calibration error, Brier score), stability and the effect of removing feature groups.

We also look at how stable the model's explanations are across five different random seeds, giving a clearer sense of how reliable these results are in practice. Furthermore, unlike prior approaches [33] relying on post-hoc explanation methods such as SHAP and LIME, our approach integrates interpretability directly into the model architecture through hierarchical gating. To structure this evaluation, we organize our study around four groups of research questions:

RQ1: How does the HSNN compare to traditional machine learning baselines in terms of detection performance and computational efficiency, and what are the inherent trade-offs between accuracy and inference latency in ransomware detection?

RQ2: To what extent does the ranking of detection models remain stable across diverse evaluation protocols (stratified, temporal, and open set), and can uncertainty estimates be leveraged to identify novel ransomware families during deployment?

RQ3: Which behavioral modalities drive detection performance, and how does the HSNN's hierarchical gating mechanism manage feature redundancy and interpretability compared to flat architecture?

RQ4: Beyond raw accuracy, how reliable are the confidence scores produced by the models (calibration), and how consistent are the learned representations across different training seeds?

This research contributes to the study of behavioral ransomware detection in multiple aspects. We present the Hierarchical Sparse Neural Network (HSNN), an architecture informed by domain knowledge that models structured behavioral feature relationships and offers interpretable insights via learned gating. We assess HSNN in comparison to classical and neural baselines across various protocols, showing that model rankings vary across evaluation settings, thereby highlighting the

 doi:10.20944/preprints202604.1252.v1

importance of protocol-aware benchmarking. We conduct additional analysis on the role of behavioral modalities via ablation and redundancy studies and assess prediction reliability through calibration and stability across multiple random seeds. We also examine the trade-offs between detection performance and computational efficiency, as well as the ability to identify potential unseen ransomware families under open-set conditions. Finally, we provide a unified evaluation system that enables reproducible analysis while integrating calibration and modality-aware evaluation. The principal innovation of this study resides in the integration of a taxonomy-aligned hierarchical model with protocol-aware evaluation and reliability analysis within a singular repeatable framework.

The rest of the paper is organized as follows: In Section 2, we speak about the MLRan dataset, how to construct a feature taxonomy, and the four ways to evaluate it. It also talks about the HSNN architecture and its four ablations, as well as the evaluation method, metrics, and plan for carrying out the experiments. Section 3 presents experimental findings pertaining to all four research questions. In Section 4, we talk about what our findings mean for science, and in Section 5, we wrap up the study, talk about its limitations, and suggest areas for future work.

2. Research Design and Methodology

Figure 1 presents the overall experimental framework of this study. The framework has 4 layers. Layer 1 (Data and future preparation): the MLRan dataset undergoes integrity checks to ensure consistency and prevent data leakage. We then construct a deterministic feature taxonomy which enables the generation of multiple feature variants, including modality-based subsets and a decorrelated feature set derived from redundancy analysis. Layer 2 (Evaluation Protocol), the layer implements four complementary evaluation protocols: standard stratified splitting, time-aware chronological splitting, family-disjoint splitting, and open-set evaluation with unseen ransomware families to evaluate model performance. Layer 3 (Model suite): In this layer we implemented a combination classical machine learning methods (e.g., SVM, logistic regression, ensemble models) and neural architectures (FlatMLP, BranchOnlyNet, HSNN variants) for ransomware detection. All models were evaluated under a unified pipeline across all protocol–feature combinations. Layer 4 (Evaluation dimension), where different performance and efficacy metric were measured to answer the study's 4 research questions. This covered predictive performance, protocol sensitivity, feature importance, redundancy, interpretability, calibration, stability, efficiency, and failure modes, providing a comprehensive evaluation of behavioral ransomware detection systems. Table 1 summarizes how each research question was operationalized across evaluation protocols, feature variants, and model classes.

Table 1. Mapping Research Questions to Experimental Design.

RQ	Analysis Focus	Protocol(s)	Feature Variants	Models
RQ1	Predictive Power & Operational Efficiency	P1–P4	full	All Models
RQ2	Robustness & Generalization	P1–P4	full	All Models
RQ3	Feature Architecture & Interpretability	P1	full, decorrelated, no_string, no_signature, behavior_only, api_registry_only	All Models (primarily HSNN)
RQ4	Reliability, Calibration, & Stability	P1–P4	full	All Models

P1: provided_or_stratified (standard i.i.d.), P2: time_aware (temporal), P3: family_disjoint (cross-family), P4: open_set_family.

 doi:10.20944/preprints202604.1252.v1

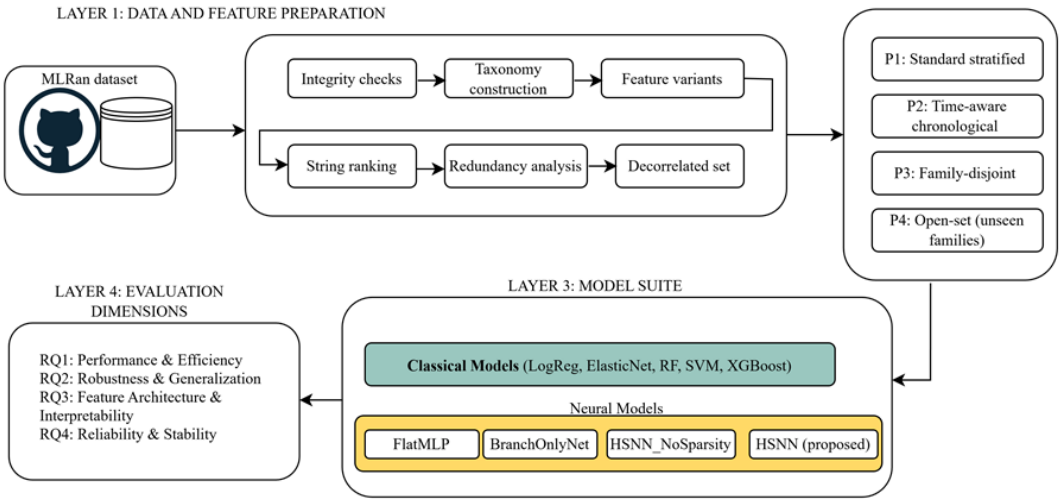


Figure 1. Study framework overview**.** An end-to-end pipeline that shows how to preprocess data, build a taxonomy, make feature variants, analyze redundancy, test under four protocols ($P_1 - P_4$), and compare classical and neural models. The framework methodically tackles RQ1–RQ4, encompassing performance, stability, interpretability, and deployment factors.

2.1. Dataset and Raw Artifacts

In this study the main dataset used was the MLRan dataset introduced by Onwuegbuche et al. [33]. It is a behavioral ransomware benchmark data with 4,880 samples (2,330 ransomware across 64 families and 2,550 goodware). The dataset had four ransomware categories including locker, crypto, RaaS, and modern variants. It has 483 distinguishing features based on dynamic sandbox analysis across nine behavioral modalities: API calls, registry operations, filesystem events, system calls, signatures, dropped artifacts, strings, network activity, and directory changes. We built our experimental pipeline using five raw artifacts from the MLRan dataset [33]: (i) training and test feature matrices, (ii) a unified label file with annotations for sample type, family, and category, (iii) dataset-level metadata, (iv) predefined training and test sample identifier partitions, and (v) a feature dictionary that connects feature indices to understandable behavioral descriptors (like API calls, registry operations, and file-system activities). All of these artifacts work together to make it possible to fully reproduce the mapping from raw behavioral signals to structured feature representations and multi-level labels used in our experiments. We checked the integrity of the data before modeling to make sure that (i) each split had its own unique sample ID, (ii) the train and test sets were completely separate, and (iii) the feature columns and dictionary keys were consistent. Any raised violations were quickly dealt with to stop leaks, which are a known cause of overly optimistic performance in machine learning pipelines [34,35]. To ensure efficient memory use and consistent numerical handling across classical and neural models, feature matrices were cast to 32-bit floating point. This preprocessing choice is consistent with prior practice while supporting controlled evaluation.

2.2. Evaluation Protocols

One of the key contributions of this study is the systematic evaluation of model behavior under four distinct protocols $(P_1, P_2, P_3 \; and \; P_4)$, motivated by the risk that independent and identically distributed (*i.i.d.*) stratified evaluations can overestimate real-world performance [36–38].

- **Stratified split (P_1):** Under this protocol, we split the original MLRan training dataset deterministically (80/20) using stratified sampling with random_state = 42. The original MLRan test set remained unchanged.

 doi:10.20944/preprints202604.1252.v1

- **Time-aware split (P_2):** In this setting, the new_timestamp feature was used to sort the combined MLRan dataset chronologically. It was then divided into training (64%), validation (16%), and test (20%) sets.
- **Family-disjoint split (P_3):** Using a fixed random seed, ransomware families with ≥ 10 samples were split into training (60%), validation (20%), and test (20%) sets. Benign samples ("goodware") were shared proportionally. This protocol evaluates generalization to previously unseen families.
- **Open-set evaluation (P_4):** In this protocol, we withheld a subset (20%) of ransomware families entirely from training; these families appear only at test time. Known families follow a 64/16/20 split, while unknown families are added to the test set and labeled accordingly. Open-set detection was measured using the complement of the maximum softmax probability, a widely adopted baseline for uncertainty-based open-set recognition [39].

We use a set of metrics to evaluate model performance across a variety of scenarios. Macro-F1 is used as the primary metric to compare detection performance across models and splits on this task. Secondary metrics include AUROCs for evaluating performance at distinguishing between positive and negative classes in open set scenarios as well as scenarios with shifting data distributions. For calibration metrics, we use the Brier score and the Expected Calibration Error (ECE). All splits are included in a single structure and we ensure that for any two splits, none of the sample_id values overlap pairwise. This is to prevent overfit to another split via data reuse. All experiments were conducted on the pre-defined splits.

2.3. Feature Taxonomy Construction

The original MLRan 483 RFE-selected features were human-readable (for example, API:NtAllocateVirtualMemory, REG: HKLM...\Run, STRING: kernel32.dll), and the prefixes directly encode behavioral modality. This allows for a deterministic, rule-based taxonomy that doesn't have any learned parts. This is similar to previous works [23–26] that used behavioral artifacts from dynamic malware analysis to create structured behavioral feature representation. Branch assignment was done by matching prefixes like API, REGISTRY, FILESYSTEM (FILE/DIRECTORY), SYSTEM, SIGNATURE, DROP_ARTIFACT, and STRING. Any features that didn't match were put in the OTHER category. To achieve this, the keyword rules for feature strings were priority-ordered: cross-branch rules were followed by branch-specific rules to group up residual features. For example, "writeprocessmemory" became "injection_memory", "http" became "network_communication", "createprocess" became "process_thread_control", "antivm" became "anti_analysis", and "service" became "service_control". Furthermore, "REGISTRY" groups "persistence", "environment_info", and "system_config". If a group consisted of less than five features, we assigned them to the corresponding fallback groups within the branch scope, for example "api_misc" and "registry_misc". The resulting taxonomy has 8 branches and is changeable. Algorithm 1: Taxonomy of the features developed in this research.

Algorithm 1: Deterministic Feature Taxonomy Construction

Input:

F: Set of selected feature names

P: Prefix-to-branch mapping

$\mathcal{R}_{global}$: Ordered cross-branch rule set

$\mathcal{R}_{branch}$: Branch-specific rule sets

τ: Minimum allowable group size

Output:

$\mathcal{T}$: Feature taxonomy

 doi:10.20944/preprints202604.1252.v1

1: Initialize $\mathcal{T} \leftarrow \varnothing$
2: **for each** feature $f \in F$ **do**
3: $b \leftarrow \text{InferBranch}(f, P)$
4: $g \leftarrow \text{InferGroup}(f, b, \mathcal{R}_{global}, \mathcal{R}_{branch})$
5: Assign f to (b, g) in $\mathcal{T}$
6: **end for**
7: **for each** branch $b \in \mathcal{T}$ **do**
8: Compute frequency N_g for each group $g \in b$
9: **for each** group $g \in b$ **do**
10: **if** $N_g < \tau$ **then**
11: Reassign all $f \in g$ to fallback group b_{misc}
12: **end if**
13: **end for**
14: **end for**
15: **return** $\mathcal{T}$
Function InferBranch (f, P):
16: **for each** $(prefix, branch) \in P$ **do**
17: **if** f starts with $prefix$ **then return** $branch$
18: **end for**
19: **return** OTHER
Function InferGroup$(f, b, \mathcal{R}_{global}, \mathcal{R}_{branch})$:
20: **for each** $rule \in \mathcal{R}_{global}$ **do**
21: **if** f matches $rule$ **then return** $group_{rule}$
22: **end for**
23: **for each** $rule \in \mathcal{R}_{branch}[b]$ **do**
24: **if** f matches $rule$ **then return** $group_{rule}$
25: **end for**
26: **return** b_{misc}

The taxonomy construction process has a computational complexity of $O(|F| \cdot (|R_{global}| + |R_{branch}|))$, because each feature is checked against both global and branch-specific rule sets.

2.4. String Feature Selection

To address the challenge of high cardinality and potential overfitting risk [40] posed by string artifacts to model generalizability, we treated them as a special case in our pipeline. Instead of either including all STRING features or removing all of them randomly, we introduced two modes of treatment. In the "class_gap" mode (default mode), we rank STRING features by the absolute difference of the mean activation of two classes on the training set (Equation (1)). This aligns with filter-based feature ranking methodologies commonly employed in high-dimensional learning contexts, especially in filter-based feature selection techniques [41]. We then select the top topk_string_features = 80 features and remove the rest.

Next, in variance mode, features are ranked by training-set variance. The ranking is always computed on training data before any model fitting. This ensured no leakage from validation or test sets into the feature selection process.

$$\text{score}(f) = |\bar{x}_{f,\text{ransomware}} - \bar{x}_{f,\text{goodware}}| \quad (1)$$

 doi:10.20944/preprints202604.1252.v1

2.5. Redundancy Analysis and Decorrelated Feature Variant

We characterize redundancy in the 483-feature space by computing the Pearson correlation matrix on the training set, and identifying features with absolute correlation greater than 0.90, following established practices for mitigating multicollinearity in high-dimensional feature spaces [41,42]. The second feature (upper-triangular column) was taken out of each pair, leaving a decorrelated feature variant. To get a better idea of redundancy, we mapped correlated pairs to their branch assignments and counted how many times they happened within and across branches. A branch-level correlation heatmap is constructed by averaging feature activations within each branch and correlating the resulting branch-level representations. We define eight feature set variants derived from the full 483-feature space: full, no_string, no_signature, no_string_no_signature, api_registry_only, behavior_only, artifact_heavy, and decorrelated (constructed by removing highly correlated features). Each variant isolates certain behavioral or artifact-driven modalities to facilitate controlled ablation analysis. Each (protocol, model) combination is used to test all variants on their own. The apply_string_mode function adds another STRING pruning layer. It has three options: all (no pruning), topk (the top 80 ranked STRING features), and drop (complete removal). All subsets of features are made using only indices from the training set, which makes sure that no information leaks from the validation or test data.

2.6. Classical Baseline Models

We adopted seven classical models, and for each combination of protocol, feature variant, and string mode, we create a new instance of each one to make sure that no state leaks between experimental conditions.

1. Dummy classifier: A most-frequent predictor that establishes a lower-bound baseline for all metrics.
2. Logistic regression (LogReg): A regularized linear model that serves as a strong and interpretable baseline for binary classification.
3. ElasticNet logistic regression (ElasticNetLogReg): A penalized linear model that combines L1 and L2 regularization, enabling implicit feature selection while retaining stability.
4. Random forest (RandomForest): An ensemble of decision trees that captures nonlinear relationships and feature interactions through bagging [43].
5. Sparse linear SVM (SparseLinearSVM): A linear support vector machine with calibrated probability outputs, included to assess margin-based classification performance.
6. XGBoost (XGBoost): A widely used gradient boosting method known for strong predictive performance and robustness across tabular datasets [44].

All classical models are trained on the training partition of each protocol split and tested on both the validation and test sets. Time is used to measure training time in wall-clock time. The average of three independent prediction passes is used to report inference time, which is done to make sure it is stable. To get an idea of the model size, each fitted model is serialized and its file size in megabytes is recorded.

2.7. Neural Model Suite

We train four different neural architectures in PyTorch [45] and compare their performance with the performance of their classical counterparts under the same settings. The training code for all networks is the same, and the evaluation criteria are the same.

2.7.1. Baseline Neural Models

Two baseline neural models were implemented.

1. FlatMLP: The flat multi-layer perceptron sees the whole feature vector as an unstructured input and doesn't use any behavioral taxonomy. It has a feed-forward structure with many fully

 doi:10.20944/preprints202604.1252.v1

connected layers, nonlinear activations, and dropout regularization. This model is a starting point for figuring out if explicitly encoding behavioral structure is better than a regular neural classifier.

2. BranchOnlyNet: BranchOnlyNet divides features into behavioral branches, which gives it a rough idea of the structure, but it doesn't model finer group-level relationships or learned gating. A separate encoder processes each branch, and the resulting representations are combined and sent to a shared classifier. This design separates the effect of branch-level modality partitioning from a flat representation.

2.7.2. Proposed Model

This section presents the design of the propose HSNN. In this work, sparsity refers to structured regularization of feature importance rather than hard feature elimination, encouraging selective weighting across behavioral groups.

2.7.2.1. Hierarchical Sparse Net (HSNN)

The main model suggested in this study is the Hierarchical Sparse Neural Network (HSNN). It directly encodes the behavioral taxonomy into the model architecture using a two-level hierarchy with learnable scalar gates. This makes it possible to choose features in a structured way and learn representations that can be understood. Figure 2 shows a schematic diagram of HSNN.

Group-level encoders: A "Group Encoder" (see Equation (2)) processes the subset of features that belong to each behavioral group (g). where x_g represents the subvector of input features belonging to group g, and $W_g$and $b_g$are learnable parameters. The scalar $l_g \in \mathbb{R}$is a learnable gate logit, and $\sigma(\cdot)$ refer to the sigmoid function. The temperature parameter (τ) controls how sharp the gating function is. Lower values make near-binary gate activations more likely. This formulation lets each group add to the downstream representation in a way that is proportional to their size, and it also lets the model ignore behavioral signals that aren't needed or are too similar to others.

$$h_g = \mathrm{ReLU}(W_g x_g + b_g) \cdot \sigma\left(\frac{l_g}{\tau}\right) \quad (2)$$

Branch-level encoders: For each behavioral branch b, the outputs of all group encoders within that branch are aggregated as defined in Equation (3). where $[h_g]_{g\in b}$ denotes the concatenation of group representations within branch b, and $l_b$is a learnable branch-level gate logit. This mechanism allows the model to control the contribution of entire behavioral modalities.

$$h_b = \mathrm{ReLU}\left(W_b\,[h_g]_{g\in b} + b_b\right) \cdot \sigma\left(\frac{l_b}{\tau}\right) \quad (3)$$

Classifier: The representations at the branch level are combined and run through a feed-forward classifier to make the final prediction (see Equation (4)). where $f_{\mathrm{MLP}}(\cdot)$ refers to a multi-layer perceptron.

$$\hat{y} = f_{\mathrm{MLP}}([h_b]_{b\in B}) \quad (4)$$

Structured sparsity regularization: We added structured sparsity to HSNN by using gate and weight penalties to encourage selective and understandable representations. As shown in Equation (5), the total training loss is defined. where $\mathcal{L}_{\mathrm{CE}}$is the class-weighted cross-entropy loss, and $p(\cdot)$is a gate regularization function. When using binary gate regularization, $p(v) = v(1-v)$, which is maximized at $v = 0.5$and minimized at $v \in \{0,1\}$, encouraging gates to converge toward discrete on/off states. Alternatively, an L1 penalty $p(v) = \mid v \mid$can be used.

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \lambda_g \frac{1}{\mid G \mid}\sum_{g\in G} p(g_g) + \lambda_b \frac{1}{\mid B \mid}\sum_{b\in B} p(g_b) + \lambda_l \frac{1}{\mid G \mid}\sum_{g\in G} \parallel W_g \parallel_1 \quad (5)$$

Domain-informed sparsity: To incorporate domain knowledge, the sparsity penalty for groups within the STRING modality is scaled by a multiplicative factor, $p_{\mathrm{STRING}}(v) = \alpha_{\mathrm{string}} \cdot p(v)$,

 doi:10.20944/preprints202604.1252.v1

where $\alpha_{\text{string}} > 1$. This reflects the higher risk of spurious correlations in high-cardinality string artifacts and biases the model toward more stable behavioral signals.

Interpretability via gates: After training, the learned gate values $\sigma(l_g/\tau)$ and $\sigma(l_b/\tau)$ are extracted and analyzed. These values give a direct measure of how important behavioral groups and modalities are, which makes it possible to understand model decisions and check for stability across random initializations.

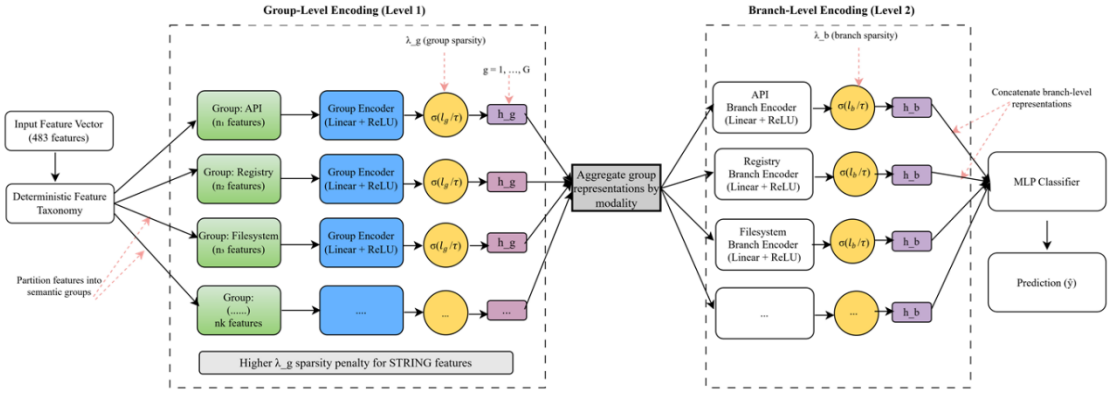


Figure 2. Architecture of a Hierarchical Sparse Neural Network (HSNN). The model encodes features using a two-level hierarchy of group-level and branch-level encoders with gates that can be learned. Structured sparsity is used at both levels, which allows for selective activation of behavioral groups and modalities while also making it easier to understand.

2.7.2.2. HSNN No Sparsity (HSNNNS)

HSNNNS retains the full hierarchical architecture and gating mechanism of the HSNN, but we disabled all sparsity-inducing penalties. This variant isolates the effect of structured sparsity by allowing gates to remain unconstrained. Comparing this model with HSNN disentangles the contribution of hierarchical structure from that of sparsity regularization.

2.7.3. Neural Training Protocol

All neural models go through the same training process. Class-weighted cross-entropy loss is used to fix class imbalance. The weights are based on the distribution of the training data and are only used during optimization. Model parameters are optimized using AdamW [46] with gradient clipping, and learning rates were scheduled using cosine annealing [46]. Early stopping [47] was applied based on validation macro F1 to prevent overfitting, and the best-performing checkpoint is restored prior to evaluation. Mini-batch training was performed using shuffled training loaders and deterministic validation and test loaders. Labels are encoded as integers, and mappings are kept the same so that decoding is always the same. Wall-clock timing over the whole training loop is used to measure how long it takes to train. Inference speed is the average of several forward passes to make sure it stays stable, and throughput is measured in samples per second. For neural models, serialized model size was not reported due to differences in storage formats relative to classical models and is excluded from efficiency comparisons. Gate values are logged from time to time during training to keep track of how they change. This lets us look at whether gates move toward sparse, stable configurations or stay spread out, which helps with both interpretability and stability assessments.

2.8. Hyperparameter Optimization and Unified Experiment Execution

To avoid overfitting and keep the computational cost reasonable, hyperparameter tuning was performed on the full feature version of each model, on the reference protocol using the Optuna framework [48]. All hyperparameter tuning was performed solely on the training set of the reference

 doi:10.20944/preprints202604.1252.v1

protocol and never on the validation or test sets. For neural models, each Optuna trial was trained for 20 epochs, and the best-found hyperparameters were then retrained on the model for 30 additional epochs for final runs reported in results. These hyperparameters were then reused for all other variations of protocols and features, to control the effect of both protocol and features independently. We trained and tested all models using the same search procedure and ran each model for 50 runs evaluating performance. Whilst some tiny gains might possibly be obtained by tuning per condition, our primary goal was to fairly compare the models in a realistic deployment scenario, where re-tuning is not always practical.

In the tables below we present the hyperparameters that we finally settled on for training the models. All experiments were run through a single pipeline. This pipeline iterates over all possible combinations of evaluation protocol, feature variant, and string processing. We run each of these iterations independently so that if one run fails, it won't bring the whole experiment down. The results of each experiment are then recorded and presented below in a structured format. In each table, the first column is the hyperparameters used to train that model, the second column is the protocol used for that model, the third column is the feature variant used for that model, the fourth column is the runtime of the model, and the fifth column details the results of the evaluation. Table 2 below details the results of the experiments. All outputs have been fully generated and tested. The outputs have been cached in memory to improve computer processing time and simply re-used as required as opposed to being remade. The execution plan allows for easy testing in all scenarios in a fully controlled and repeatable manner.

Table 2. Best hyperparameters identified via Optuna (50 trials per model). All reported configurations override default initialization settings.

Model	Best Value	Key Hyperparameters
HierarchicalSparseNet (HSNN)	0.99615	lr=0.00511, weight_decay=4.53e-4, dropout=0.482, hidden_dim_group=64, hidden_dim_branch=32, λ_g=1.02e-3, λ_b=1.00e-4, λ_input=2.27e-6, τ=0.0796, STRING multiplier=4.36
BranchOnlyNet	0.99615	lr=0.00734, weight_decay=6.55e-4, dropout=0.411, hidden_dim_group=32, hidden_dim_branch=64
HSNN (No Sparsity)	0.99615	lr=0.00702, weight_decay=1.57e-4, dropout=0.354, hidden_dim_group=64, hidden_dim_branch=32
FlatMLP	0.99615	lr=0.00849, weight_decay=1.04e-3, dropout=0.390, hidden_dim=64
Sparse Linear SVM	0.99358	C=3.68
Random Forest	0.99230	n_estimators=300, max_depth=None, min_samples_leaf=1
ElasticNet LogReg	0.99230	C=5.76, l1_ratio=0.392
Logistic Regression	0.99101	C=6.35
XGBoost	0.98974	n_estimators=500, lr=0.0104, max_depth=10, subsample=0.656, colsample=0.504, α=1.49, λ=1.58

The Optuna optimizer's output shows that all of the neural architectures reached about the same level of performance at their best. This means that just looking at the predictive accuracy doesn't tell us which model works better with this dataset. The proposed HSNN is evaluated based on performance, interpretability, stability, and robustness, with its structured design providing additional advantages.

 doi:10.20944/preprints202604.1252.v1

2.9. Evaluation Metrics

We used different scikit-learn [49] classification evaluation metrics library to evaluates each model's predictions. These include accuracy and balanced accuracy, which takes into account class imbalance and averages recall for each class. Macro-averaged precision, recall, and F1-score were presented, considering all classes uniformly irrespective of their support. We used ROC-AUC to measure discrimination performance. This is the area under the receiver operating characteristic curve for binary classification and the macro one-vs-rest AUC for multiclass settings. We used log-loss and the Brier score (as shown in Equation (6)) to measure how well the model did at predicting probabilities. We also used Expected Calibration Error (ECE) to measure the quality of the calibration. This was done over B =10 uniform bins that covered [0,1] as shown in Equation (7).

$$\mathrm{BS} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2 \quad (6)$$

$$\mathrm{ECE} = \sum_{b=1}^{B}\frac{|\,S_b\,|}{N}\;|\,\mathrm{acc}(S_b) - \mathrm{conf}(S_b)\,| \quad (7)$$

where S_b denotes the set of samples whose predicted positive-class probability falls within bin b, $\mathrm{acc}(S_b)$ is the fraction of positive samples in that bin, and $\mathrm{conf}(S_b)$ is the mean predicted probability in that bin [50]. Log-loss, Brier score, and ECE are computed only for binary classification tasks with available probability estimates and are recorded as NaN for models without probability outputs.

This study emphasizes comparative trends across protocols and feature configurations. Because the reported results are primarily aggregated across evaluation settings, formal hypothesis testing is left outside the current scope and is identified as a direction for future work.

2.10. Seed Stability Analysis

To assess seed impact on models performance, each neural model was trained with five different random seeds (7, 21, 42, 99, 137). Before initializing the model, all relevant random states are reset for each run. Training then continues until convergence, using the same budget and early stopping criteria as the main experiments. For models that use gates, the mean and standard deviation of gate values across seeds are used to measure stability. The coefficient of variation as defined in Equation (8) was used to normalize variability. where $\bar{g}$and $\sigma_g$denote the mean and standard deviation of gate values for group g. Rank stability was further assessed using Jaccard similarity between the top-k most active groups across seeds (see Equation (9)).

$$\mathrm{CoV} = \frac{\sigma_g}{\bar{g}} \quad (8)$$

$$J(S_i, S_j) = \frac{|\,S_i \cap S_j\,|}{|\,S_i \cup S_j\,|} \quad (9)$$

The average and variance of this measure for all seed pairs show how consistent learned behavioral importance is. For HSNN, gate values were also monitored during training to examine convergence behavior and ascertain whether sparsity patterns stabilize or vary over time.

2.11. Calibration and Robustness Analysis

In this work we investigate the calibration quality of the predicted probabilities for models that output probabilistic predictions. For this purpose, standard calibration methods have been applied to generate so-called reliability diagrams [50]. In addition to the reliability diagrams, for each model so called confidence distribution plots have been generated, which show the differences of the prediction confidences for correct and incorrect classifications. Additionally, the expected Calibration Error (ECE) and the Brier scores have been calculated to get a quantitative statement about the

 doi:10.20944/preprints202604.1252.v1

calibration quality. We present a method to compare the probabilistic reliability of models. We also present a method to measure robustness. Robustness is often measured during training by applying a form of regularization in order to improve generalization. In this work, robustness is measured in a fixed test set fashion, by applying controlled perturbations to the input of a network without retraining. We examine class probability by measuring the expected likelihood over all possible dropout masks, by adding noise to the input features (feature corruption), and by removing entire behavioral branches (modality removal). We also show how the models perform when perturbed to different extents, and use this to generate degradation curves and modality-level impact heatmaps to visualize model sensitivity and understand the key behavioral components affected. These additional metrics give further insight into the probabilistic reliability and robustness of the models when subject to distributional perturbations. We name these additional metrics Macro F1 scores.

2.12. Open-Set Detection and Uncertainty Estimation

To evaluate model behavior under unseen conditions, we adopt an open-set evaluation protocol (P4), where a subset of ransomware families is excluded from training and introduced only at test time. Uncertainty is estimated using the complement of the maximum predicted probability, as defined in Equation (10), where p_c denotes the predicted probability for class c. Higher values indicate lower confidence and potential novelty. Open-set performance is evaluated using AUROC, which measures the model's ability to distinguish known from unknown samples.

$$s(x) = 1 - \max_c p_c \quad (10)$$

2.13. Reproducibility Controls

Experiments are run on a Lenovo 83LY Lenovo laptop running Windows 11 (64-bit) with an Intel Core i7-14700HX processor, 64 GB RAM, 1 TB NVMe SSD and an NVIDIA GeForce RTX 5060 Laptop GPU. We primarily use Python 3.13.12 with several scikit-learn, NumPy and pandas dependencies. All the used packages, including the versions, are written to disk at runtime. Furthermore, all Python packages with non-deterministic states are pinned to a certain value, such as the pseudo-random number generators provided by Python, NumPy and PyTorch, and GPU settings. Finally, all the experiments outputs are given in the same format, and all the experiment settings are managed through a single, central place to avoid any drift. Neural hyperparameter search used 50 Optuna trials with a 20-epoch trial budget; final models were retrained for 30 epochs using the selected configuration.

3. Results and Discussion

This section summarizes the results we obtained when experimenting on the specific cases related to each research question posed in Section 1. Table 1 presents the mapping between research questions, the evaluation scenario (i.e., the set of protocols for each pair of models), feature variants and model classes. We detail one research question per subsection, which allows each experimental results to mapped to the specific question it addresses. We report macro-F1 as the primary metric for model comparison, with AUROC and calibration metrics providing complementary insights.

Preliminary Result: Feature Taxonomy Construction

Before turning to RQ1–RQ4, we briefly validate the constructed feature taxonomy used by the hierarchical models. Table 3 shows how the 483 RFE-selected features are distributed across the seven behavioral modalities defined by the deterministic taxonomy in Section 2.3. The distribution closely matches the feature statistics reported in the MLRan dataset study [33], confirming that our feature-selection pipeline was successfully reproduced. The STRING branch is the largest, comprising 46.2% of selected features (223), followed by REGISTRY with 18.6% (90) and API with 13.0% (63). The NET branch contributes no features after RFE, suggesting network-level indicators were not individually discriminative for this feature set. After merging (τ = 5), the 483 features are organized into 21 functional groups across the seven branches, forming the two-level hierarchy used by the

 doi:10.20944/preprints202604.1252.v1

Hierarchical Sparse Net's Group Encoder and Branch Encoder. Figure 3 provides an overview of the taxonomy.

Table 3. Distribution of RFE-Selected Features Across Behavioral Branches.

Branch	Features	% of Total	Groups
STRING	223	46.2	4
REGISTRY	90	18.6	4
API	63	13.0	4
SYSTEM	38	7.9	2
FILESYSTEM	26	5.4	3
DROP_ARTIFACT	23	4.8	2
SIGNATURE	20	4.1	2
Total	483	100.0	21

Note: Group counts reflect post-merge values with merge threshold $\tau = 5$.

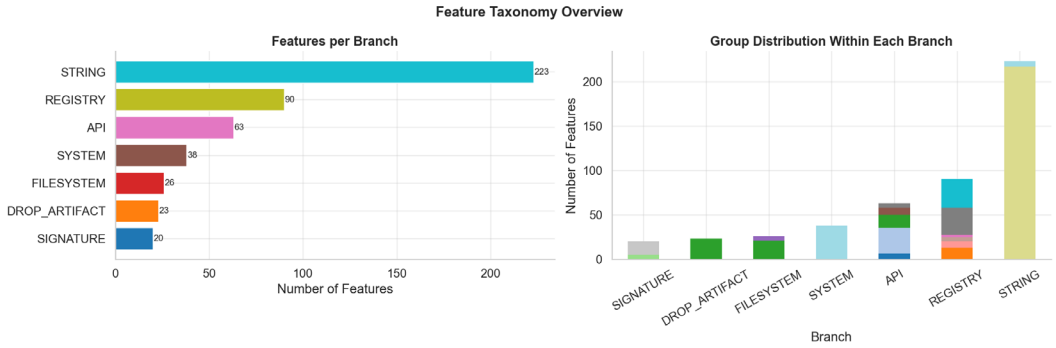


Figure 3. Feature Taxonomy Overview.

3.1. RQ1: Predictive Power & Operational Efficiency

3.1.1. Training Dynamics and Convergence

Before showing predictive results for the two models, we present the training performance to have a deeper understanding of the convergence. In the figure above, we show the training convergence of HierarchicalSparseNet (HSNN) and FlatMLP (baseline neural network) trained with the best hyperparameters found after hyperparameter search. Although the two models achieved high F1 macro and balanced accuracy (0.93 and 0.93 respectively), there was a significant difference in the convergence behaviors of the two models. HierarchicalSparseNet achieves stable and smooth performance peak at Epoch 17 with small variations between training and validation performance. Even though FlatMLP also achieved stable performance, there were local variations between Epoch 7 to 10; then it reached stable performance in the latter epochs. The training and validation curves for the deep network (HSNN) are more smoothly aligned than those for the flat network (FlatMLP), indicating that learning is more stable than that of FlatMLP and is marked by large oscillations.

3.1.2. Predictive Performance Across Protocols

Table 4 shows the average model performance, calibration, and complexity across protocols (full features only). We evaluated the performance of the proposed Hierarchical Sparse Net (HSNN) against a set of classical and modern machine learning baselines. Performance was aggregated across all evaluation protocols using the full feature set to ensure a fair comparison. In addition to predictive performance, we report calibration and architectural complexity to provide a more comprehensive

 doi:10.20944/preprints202604.1252.v1

view of deployment-relevant characteristics. Table 4 summarizes the results. We observed that ransomware detection on the MLRan [33] dataset is consistently high across all evaluated models, with average macro F1 scores exceeding 97%. Among the evaluated models, FlatMLP achieved the highest average F1-macro score (0.9860). Next was the proposed HSNN model with an average F1-macro of 0.9839, representing a marginal difference of approximately 0.21% compared to FlatMLP. The logistic regression model obtained comparable performance (0.9835). However, ensemble methods; XGBoost (0.9776) and Random Forest (0.9775) achieved lower average scores compared to all other models. Although FlatMLP attained the highest nominal classification performance, the margin between it and HSNN remains small. Hence, HSNN maintains competitive performance across protocols, with relatively small absolute variation despite shifts in model ranking.

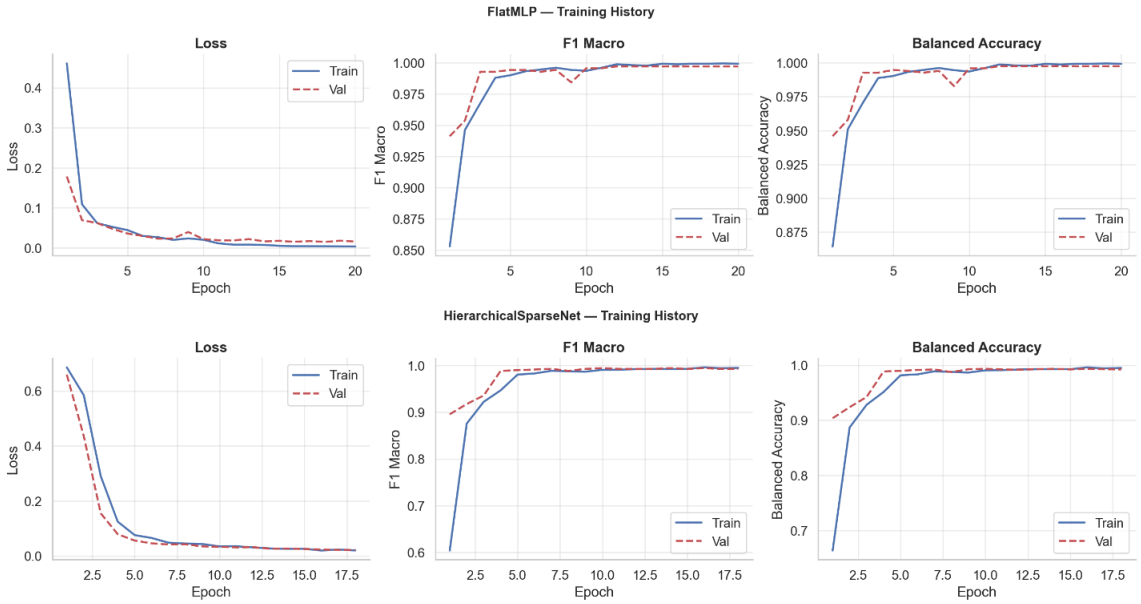


Figure 4. Training convergence for the best results (best hyperparameters found during hyperparameter tuning) for the FlatMLP and HierarchicalSparseNet (HSNN) neural networks. The hyperparameter tuning was performed training each model for 20 epochs, and then the best hyperparameters were used to train the models from scratch for 30 epochs.

Table 4. Average Model Performance, Calibration, and Complexity Across Protocols (Full Features Only).

Model	F1-Macro (Avg) ↑	ECE (Avg) ↓	Brier Score (Avg) ↓	Parameters
FlatMLP	0.9860	0.0164	0.0142	35,266
Hierarchical Sparse Net (HSNN)	0.9839	0.0108	0.0122	20,430
LogReg	0.9835	0.0115	0.0125	–
XGBoost	0.9776	0.0131	0.0176	–
Random Forest	0.9775	0.0411	0.0219	–

The table displays accuracy (percent correct) and calibrated accuracy (error when reporting confidence) for the models. Table 4 describes the expected error of each model for reporting confidence. The best calibrated model was the HSNN model at 0.0108, compared to 0.0164, 0.0131 and 0.0115 for FlatMLP, XGBoost and Logistic regression respectively. But the Random Forest model had a higher calibration error of 0.0411. Also, HSNN had 20,430 parameters, while FlatMLP had 35,266, which is about 42% less. This shows that HSNN was simpler. Notably, HSNN maintains performance comparable to the top-performing models despite its lower parameter count. So,

 doi:10.20944/preprints202604.1252.v1

FlatMLP has the best average detection performance, while HSNN is a good alternative with a smaller model size and less calibration error.

3.1.3. Operational Efficiency

We compare the models based on their classification performance, inference latency and model size. Table 5 shows the performance for all model when using the full set of features. We can see that the FlatMLP is leading in terms of classification performance but at the cost of a larger model size. The proposed HSNN reduces the parameter count by approximately 42%, while at the same time keeping the performance close to the one of the FlatMLP. The non-sparse version has a similar performance but does not bring much of a reduction in terms of model size. Logistic regression has a very low inference latency and low computational cost but slightly lower performance than the neural networks. The tree-based models have comparable performance but higher inference latency.

Figure 5 further illustrates the trade-off between detection performance and latency where HSNN occupies a competitive accuracy–compactness position, despite higher latency than FlatMLP. Inference categories reflect relative computational cost observed during evaluation, where classical linear models exhibit the lowest latency, tree-based models moderate latency, and neural architectures higher, yet still within practical deployment bounds. Inference time (**ms**) is reported as mean ± standard deviation of per-sample prediction time, computed across all evaluation protocols using the full feature configuration. Table 5 shows that the proposed HSNN is more efficient with parameters (−42%), but it takes longer to make inferences than FlatMLP (see Figure 5). This shows a clear trade-off between how sparse the structure is and how much extra work it takes to run.

Table 5. Performance–Efficiency Trade-offs (Full Features, Averaged Across Protocols).

Model	F1-Macro	Inference (ms)	Parameters
FlatMLP	~0.986	low (0.0080 ± 0.0003)	35,266
HSNN	~0.984	higher (0.0519 ± 0.0013)	20,430
HSNN_NoSparsity	~0.984	higher (0.0475 ± 0.0012)	20,430
BranchOnlyNet	~0.982	low (0.0136 ± 0.0003)	15,866
LogReg	~0.983	very low (0.0025 ± 0.0002)	—
XGBoost	~0.978	moderate (0.0186 ± 0.0020)	—
RandomForest	~0.977	higher (0.0532 ± 0.0040)	—

See Table 6 for the number of features for each of the feature variants. The number of input dimensions ranges from 483 (all features) down to 153 input dimensions for the API/registry-only features. As can be seen from various experiments, some of the feature reduction methods (e.g., removing the string and the signature features) and decorrelation led to large reductions in the dimensionality while suffering only a small loss in performance.

In addition to the performance metrics in Table 2, which show the ability of the models to predict, accuracy and speed, we also test the robustness and the generalization capability of a model to different evaluation scenarios. The results of this test are presented in the next section.

Table 6. The number of dimensions for different feature variants. The table shows how removing features and decorrelating them can make the input size smaller.

Feature Variant	Total Features	Drop	Top-k
Full	483	260	340
Decorrelated	375	197	259
No Signature	463	240	320
No String	260	260	260

 doi:10.20944/preprints202604.1252.v1

No String + No Signature	240	240	240
Behavior Only	217	217	217
Artifact Heavy	266	43	123
API / Registry Only	153	153	153

Figure 5. Performance–latency trade-off across models. HSNN strikes a good balance between detection performance and model size, keeping accuracy close to the top while using fewer parameters. However, it has a longer inference time than FlatMLP.

3.2. RQ2: Robustness & Generalization Across Evaluation Scenarios

3.2.1. Protocol Sensitivity and Ranking Shifts

We qualitatively evaluated ranking of models based on their performance on all task protocols. We examined to what extent performance behaviors observed on the standard stratified test setup (P_1) generalize to more challenging and realistic test scenarios $(P_2 - P_4)$. In Table 7 we present macro F1 scores obtained on the full feature set for all models and all test setups. We further illustrate per protocol ranking transitions in Figure 6. The results demonstrate that, although absolute detection performance remains high ($F1 > 0.96$) across all settings, the relative ordering of models is highly sensitive to the choice of evaluation protocol.

We observed the FlatMLP model obtained highest or tied-highest rank in protocols P_1, P_2, and P_3. However, its ranking dropped to fourth in the open-set setting (P_4) as seen in Figure 6. This indicates a reduction in relative performance when it encounters an entirely new ransomware family. On the other hand, the hierarchical models (HSNN and HSNN_NoSparsity) exhibit the most consistent generalization across the more difficult protocol (P_4). In P_1, we observed that HSNN_NoSparsity ties for the top rank. This may reflect the higher capacity of its dense connections in an IID setting. Conversely, we observed the sparse HSNN ranking lower in P_1 but demonstrated excellent robustness in the temporal protocol (P_2). This may indicate that structural sparsity acted as a regularizing influence against temporal feature drift. Notably, the two models (HSNN and HSNN_NoSparsity) ranked top-1 in the most challenging scenarios (P_4). This observation suggests that hierarchical architecture plays an important role for cross-family generalization.

 doi:10.20944/preprints202604.1252.v1

Table 7. Macro F1 Scores and (Ranks) Across Evaluation Protocols.

Model	P1: Stratified	P2: Temporal	P3: Fam-Disjoint	P4: Open-Set
FlatMLP	0.9800 (1)	0.9860 (1)	0.9865 (1)	0.9913 (4)
HSNN	0.9789 (4)	0.9797 (5)	0.9838 (2)	0.9930 (1)
HSNN_NoSparsity	0.9800 (1)	0.9784 (7)	0.9838 (2)	0.9930 (1)
LogReg	0.9789 (4)	0.9808 (4)	0.9821 (4)	0.9922 (3)
ElasticNetLogReg	0.9795 (3)	0.9783 (8)	0.9749 (7)	0.9896 (6)
BranchOnlyNet	0.9774 (6)	0.9796 (6)	0.9793 (5)	0.9904 (5)
SparseLinearSVM	0.9774 (6)	0.9783 (8)	0.9785 (6)	0.9878 (7)
XGBoost	0.9738 (8)	0.9809 (3)	0.9696 (9)	0.9861 (8)
RandomForest	0.9666 (9)	0.9860 (1)	0.9713 (8)	0.9861 (8)

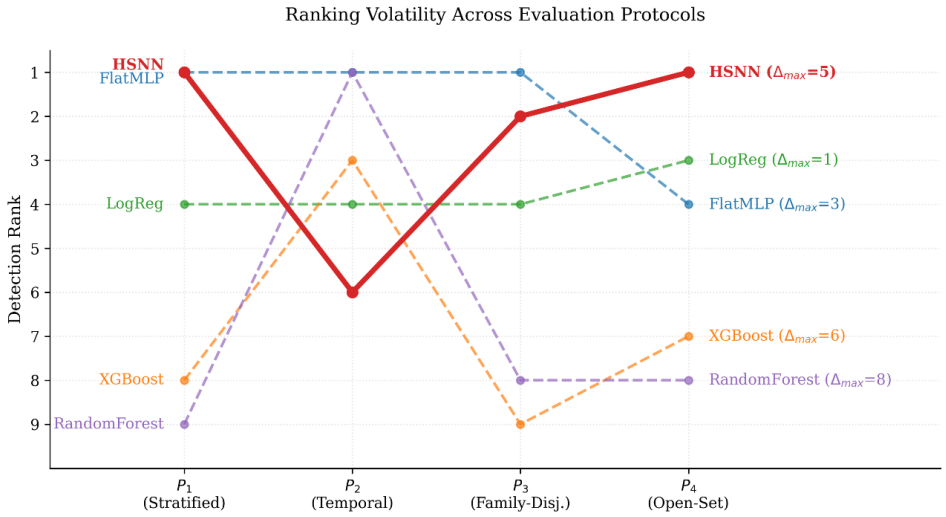


Figure 6. Model detection rank transitions across four evaluation protocols ($P_2 - P_4$). Lines illustrate the relative performance shifts of each model as the data partitioning strategy moves from stratified to open-set family detection.

However, we observed the largest ranking variability in the tree-based ensemble models. Random Forest transitions from the lowest rank in P_1 to a tied top rank in the temporal protocol (P_2), before regressing to lower ranks in P_3 and P_4. Similarly, we saw a significant improvement from the XGBoost in P_2 but ranked lower in the family-disjoint and open-set settings. These observed fluctuations are reflected in the larger maximum rank shifts of the ensemble-tree methods. This indicates that their performance is particularly sensitive to the specific data partitioning strategy and temporal distribution. In contrast, LogReg remained remarkably stable, maintaining a consistent top-4 position across all protocols. This finding suggests that while linear models may not reach the peak accuracy of deep architectures in every scenario, they might be less susceptible to the biases introduced by different partitioning strategies. In our experiments, we observed that different models got ranked differently across different protocols. In particular, HSNN got the best results in the most realistic scenario, P_4. This highlights the need for multi-protocol evaluation in detecting ransomware in non-stationary environments.

3.2.2. Detection of Unseen Ransomware Families and Uncertainty

This section investigates the detection performance of the models on the unseen families of ransomware using the open-set protocol (P_4) to analyze the calibration of the models using ECE and

 doi:10.20944/preprints202604.1252.v1

Brier score. Figures 7 shows models' performance across unseen families. Figure 8 shows a plot of models' performance across unseen families and expected calibration error. We observed that HSNN achieves one of the highest macro-F1 scores under the open-set protocol while also exhibiting lower calibration error than several competing models. This suggests that the model generalizes well within the evaluated open-set family setting and produces comparatively more reliable uncertainty estimates under this protocol. According to the results shown in the open-set setting (see Figure 7), our proposed HSNN model still holds the highest detection rate in learning hierarchical representations for unseen ransomware families. From the calibration results (see Figure 8), our proposed model has lower ECE and Brier scores than other models. It is also observed that the calibration errors of the tree-based models are higher. As shown in the open-set errors of the open-set scenario, most of the open-set errors are caused by samples with lower confidence scores, which serve as a warning indicator for potential detection errors. Hence, our proposed model not only holds the highest detection rate in the open-set scenario but also has better calibrated uncertainty estimates, which is also important for the security applications, because a false prediction with high confidence can cause severe damage.

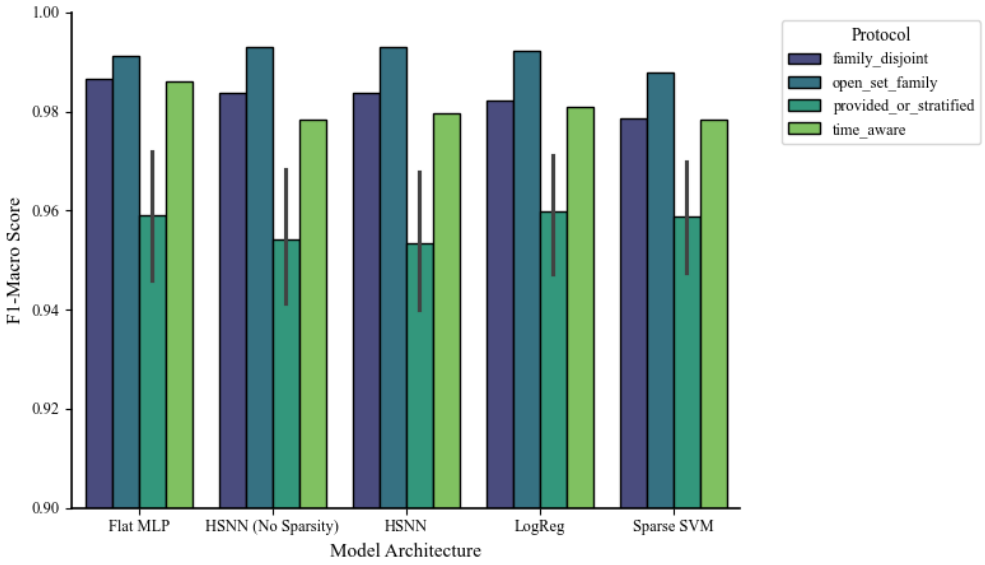
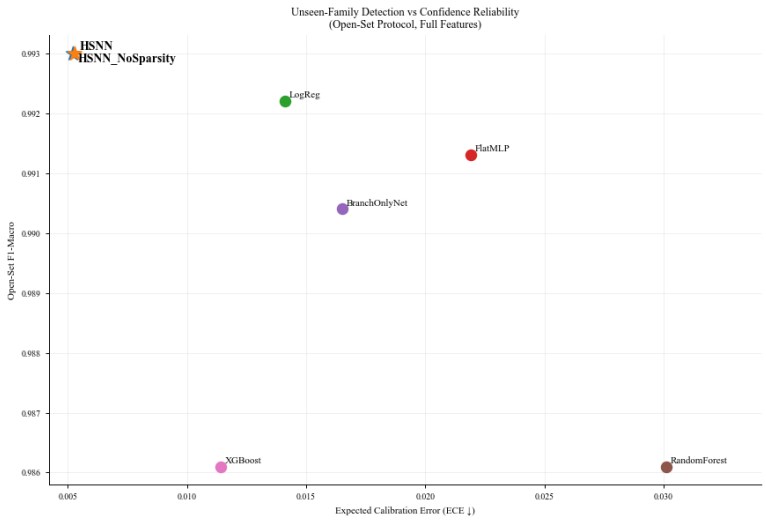


Figure 7. Macro-F1 comparison across evaluation protocols, including open-set conditions representing unseen ransomware families. Performance variation reflects distribution shift, while HSNN remains competitive, particularly under open-set scenarios, indicating robustness under uncertainty.



 doi:10.20944/preprints202604.1252.v1

Figure 8. Relationship between open-set detection performance and confidence reliability under the open-set protocol. High F1-macro and low calibration error are both attained by models oriented toward the top-left. This advantageous area is occupied by HSNN and HSNN_NoSparsity, suggesting comparatively strong open-set performance with well-calibrated uncertainty estimates under the evaluated setting.

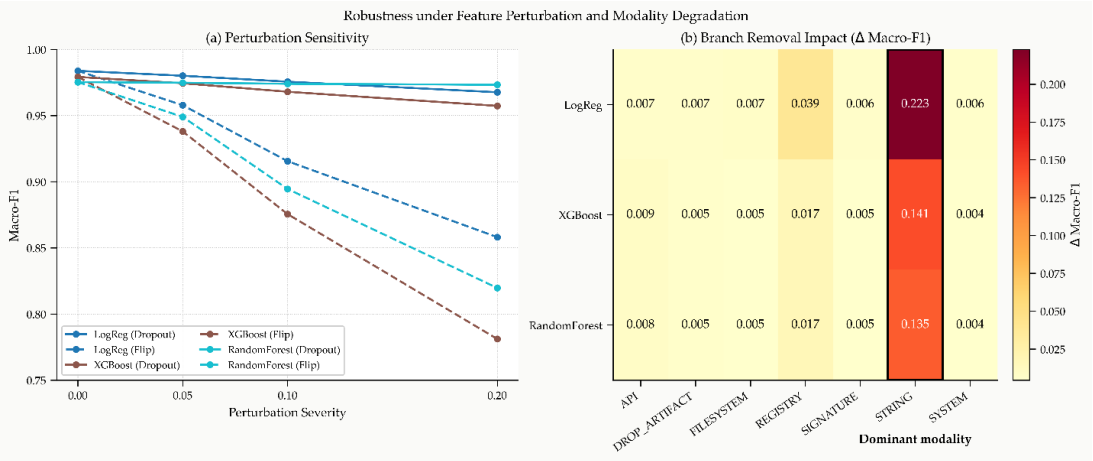
3.2.3. Robustness to Feature Perturbation and Modality Degradation

In addition to the comparisons above, we performed an additional test of baseline models under controlled perturbation and modality degradation. We measured and recorded the performance decrease of models when features were perturbed or degraded. We used two different types of perturbations: (i) feature dropout and (ii) feature flipping. For measuring the performance decrease when features were degraded to train the models, we performed the perturbation experiments at three different severities: 5%, 10%, and 20%.

This section details the robustness of all three models. Figure 9 and Table 8 illustrate the results. Random feature dropout results in a slight decrease in performance for all three models as the level of perturbation increased, however, random feature flipping results in a dramatic decrease in the models' performance. Specifically, the flippable XGBoost model dropped from 0.9792 to 0.9573 at a perturbation level of 20%, the flippable Random Forest model dropped from 0.9753 to 0.9733, and the flippable LogReg model dropped from 0.9838 to 0.9676. Further analysis shows that performance is significantly poorer when corrupt inputs are provided compared to when missing feature information is provided. This is a major limitation of the system when trying to detect objects in adverse, hostile or noisy environments. The investigation of branch removal shows that the STRING modality is the main factor that affects detection performance in all models. In every example, removing STRING features causes the most damage, with LogReg dropping by 0.2230, XGBoost dropping by 0.1412, and Random Forest dropping by 0.1349. In contrast, taking away other modalities like API, SIGNATURE, and FILESYSTEM only has a little effect on performance.

Table 8. Robustness of classical baselines under feature perturbation and modality removal. We report the baseline Macro-F1 score along with the score after 20% of the features are dropped out and 20% of the features are flipped. The worst modality column shows the feature branch that, when removed, does the most damage.

Model	Baseline F1	Dropout20 F1	Δ	Flip20 F1	Δ	Worst modality removed	F1 after removal	Δ
LogReg	0.9838	0.9676	0.0162	0.8581	0.1257	STRING	0.7608	0.2230
XGBoost	0.9792	0.9573	0.0219	0.7811	0.1980	STRING	0.8380	0.1412
RandomForest	0.9753	0.9733	0.0020	0.8197	0.1556	STRING	0.8404	0.1349



 doi:10.20944/preprints202604.1252.v1

Figure 9. Robustness of classical baselines under feature perturbation and modality degradation. (a) Macro-F1 plotted against the severity of perturbation for feature dropout and feature flipping. (b) The absolute Macro-F1 drop that happens when each behavioral branch is taken away. Removing STRING has the most effect on all models, but corruption has a bigger effect than missingness.

In general, our results suggest that baseline models can handle moderate feature sparsity well, but they are quite sensitive to feature corruption and rely heavily on STRING-based behavioral signals. This shows both the strengths and weaknesses of current ways to detect behavior and how important it is to test models in situations where they are controlled to degrade. This supplementary robustness analysis is intentionally limited to representative classical baselines to establish a reference sensitivity profile; extending the same perturbation study to the full neural model suite is left for future work.

3.3. RQ3: Feature Architecture & Interpretability

3.3.1. Feature Modality Contribution

We evaluated all models' performance under multiple feature variants, including the removal of specific features so we could analyze the contribution of different feature modalities. Figure 10(A) presents model performance under the stratified protocol (P_1) across feature variants. However, Figure 10(B) uses the entire feature set to summarize performance across evaluation protocols. The model's performance varies according to the available feature modalities, as shown in Figure 10(A). In particular, all models showed a discernible decline in performance when string-based features were eliminated. For instance, when string features were eliminated, FlatMLP dropped from roughly 0.980 under the full feature set to roughly 0.958. For HSNN and other models, a similar pattern is seen, suggesting that string-based features significantly improve detection performance.

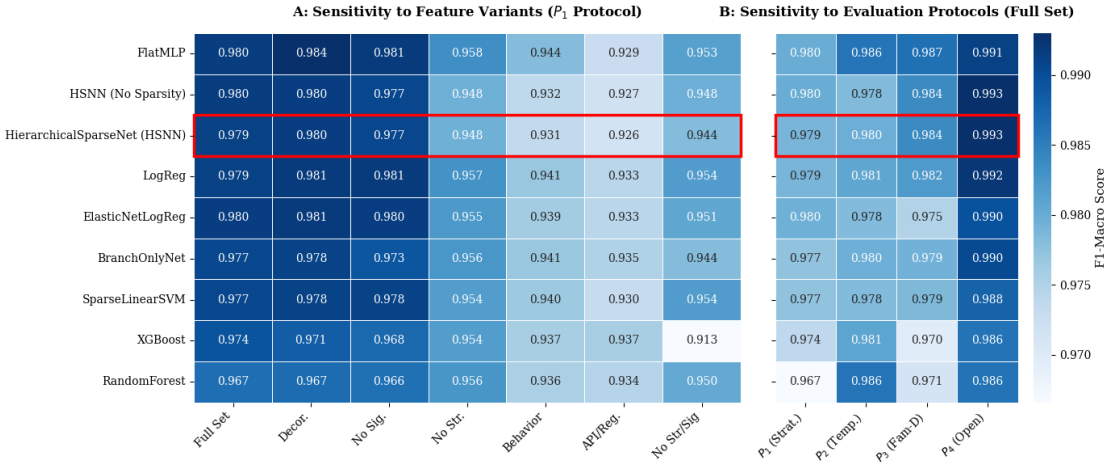


Figure 10. Model performance across feature variants (A) and evaluation protocols (B). Performance decreases under feature ablation, with string-based features showing the strongest impact.

To systematically assess the contribution of each modality, we conduct an ablation analysis by removing individual feature groups. We observed a consistent decline in performance (see Figure 10) as feature subsets were removed. This indicates that each modality contributes to ransomware detection performance. Remarkable, the string-based features exhibited the most noticeable impact, approximately 2–3% reduction in macro F1 across models (see Table 9). Their removal saw a clear reduction in the overall model's macro F1 score compared to the full feature setting. This pattern was seen to be consistent across architectures, indicating that string-derived information remains a key component of the ransomware detection signal. When models were train with behavior-only or API/registry-only features, performance decreases further across all models, typically in the range of 3–5% (see Table 9). Nevertheless, despite this reduction, the relative ordering of models was largely

 doi:10.20944/preprints202604.1252.v1

consistent. This consistency suggests that differences in performance are primarily driven by feature availability rather than model-specific sensitivity.

As seen in Figure 10(B), the proposed model (HSNN) achieved performance trends that closely track those of FlatMLP across all feature variants. Both models behaved similarly to that observed with the feature reduction experiments and the relative performance between models was preserved, without introducing observable instability due to the hierarchical structure. Figure 10(B) provides a complementary view of protocol sensitivity using the full feature set. Table 9 summarizes the relative performance reduction across feature variants. HSNN and HSNN_NoSparsity have the highest average reductions at about 3.4% while the Random Forest has the lowest at about 1.9%. Most models descended at a percentage reduction between about 2 and 3.5%, indicating broadly consistent dependence on feature modalities. We calculated the percentage reduction as, $reduction\ (\%) = \frac{F1_{full} - F1_{variant}}{F1_{full}} \times 100$.

Table 9. Relative Performance Reduction Across Feature Variants (P1, Stratified).

Model	Full	%↓ String	%↓ Signature	%↓ Behavior Only	%↓ API/Registry	Avg % Reduction
FlatMLP	0.9800	2.24%	-0.05%	3.71%	5.20%	2.78%
HSNN	0.9789	3.21%	0.15%	4.86%	5.38%	3.40%
HSNN_NoSparsity	0.9800	3.32%	0.27%	4.87%	5.39%	3.46%
LogReg	0.9789	2.26%	-0.16%	3.92%	4.66%	2.67%
ElasticNetLogReg	0.9795	2.53%	0.00%	4.18%	4.71%	2.86%
BranchOnlyNet	0.9774	2.22%	0.42%	3.78%	4.30%	2.68%
SparseLinearSVM	0.9774	2.43%	-0.10%	3.87%	4.82%	2.75%
XGBoost	0.9738	2.06%	0.57%	3.73%	3.74%	2.53%
RandomForest	0.9666	1.12%	0.05%	3.12%	3.33%	1.91%

Our experiments show that modality of features has an impact on the performance of a ransomware detection model. While the full set of features are always useful, removing a modality, such as the string modality, can lead to a substantial drop in performance. Across the evaluated feature variants, the hierarchical models remained broadly competitive with the flat baseline.

3.3.2. Redundancy Analysis

We compared models performance between the full feature set and a decorrelated variant in which highly correlated features are removed to assess redundancy in the feature space. We present our observation in Table 10, which shows the corresponding F1-macro scores under the stratified protocol (P_1). We observed a minimal difference in performance between the full and decorrelated feature sets across all models. For example, FlatMLP increases slightly from 0.9800 to 0.9836, while proposed HSNN model saw a change, from 0.9789 to 0.9795. However, XGBoost showed a small decrease from 0.9738 to 0.9712. That's, our observation shows that across all models, performance changes stayed within a narrow range of approximately −0.0026 to +0.0036. From this it can be said that removing highly correlated features does not significantly affect model performance. Moreso, the absence of a consistent increase or decrease across models suggests that redundancy exists in the feature space. However, it does not strongly influence predictive performance. Hence, our results suggest that the feature space contains redundant information, but this redundancy does not significantly impact detection performance. The decorrelated feature set provides a more compact representation while preserving comparable predictive capability.

 doi:10.20944/preprints202604.1252.v1

Table 10. Model Performance Comparison Between Full and Decorrelated Feature Sets (P1, Stratified Protocol).

Model	Full (F1)	Decorrelated (F1)	Δ (Decor – Full)
FlatMLP	0.9800	**0.9836**	**+0.0036**
LogReg	0.9789	0.9805	+0.0016
ElasticNetLogReg	0.9795	0.9805	+0.0010
BranchOnlyNet	0.9774	0.9784	+0.0010
SparseLinearSVM	0.9774	0.9784	+0.0010
HSNN	0.9789	0.9795	+0.0006
RandomForest	0.9666	0.9671	+0.0005
HSNN_NoSparsity	0.9800	0.9795	**−0.0005**
XGBoost	0.9738	0.9712	**−0.0026**

We examine pairwise feature correlations to further quantify redundancy. Strong correlations are comparatively rare in the feature space, as evidenced by the 217 feature pairs with high correlation ($|r| > 0.9$) and pair density of 0.001864. With 203 pairs classified as within-branch and only 14 as cross-branch, most of these correlations take place within the same behavioral groups. In particular, registry and string-related features account for many highly correlated pairs, with 120 and 69 pairs respectively, while other feature groups exhibit minimal redundancy. In addition, there are many exact duplicates present in both the training and test sets (684 duplicates in the training set and 286 in the test set). This does not indicate leakage since the protocol splits samples (not feature-vector unique-ness) but it does indicate that there are repeating behavioural patterns in the data which should decrease the task difficulty. However, performance on the full feature set as opposed to the performance on the data with the redundant features decorrelated does not show a decrease in performance.

Figure 11 presents the correlation structure at the branch level, computed using mean activations across feature groups. The findings demonstrate that strong correlations are mostly found between core behavioral branches, especially between registry, filesystem, and API features. For instance, filesystem and registry reach 0.78, indicating closely related behavioral patterns, whereas API and registry show a high correlation of 0.90. On the other hand, string-based features show a slight negative correlation with system-level features and consistently low correlations with other branches, with values ranging from 0.11 to 0.27. This implies that complementary information provided by string features is mostly independent of other behavioral modalities. In a similar vein, system-related characteristics exhibit moderate correlations, suggesting partial independence.

In summary we have (i) that this redundancy is not random, (ii) that it is localized to particular categories of behavior and (iii) that there is little redundancy between modalities. This result is not of variance with our previous findings that removal of highly correlated features has little effect on network performance.

3.3.3. Hierarchical Interpretability

We analyzed the proposed HSNN learned group gate values across multiple random seeds to examine the behavioral structure captured by it. Figure 12 illustrates the HSNN gate values for each behavioral group over five independent runs. The results revealed that gate values are relatively consistent across seeds for all groups. For example, the api_other group varies between approximately 0.62 and 0.70. Also, the string_misc remained within the range of 0.62 to 0.66. For each group, the variation was limited, indicating that the learned importance of each behavioral group is stable with respect to random initialization. The distribution of the gate values in Figure 13 shows that HSNN does not solely rely on a small set of features. Importance is distributed throughout the different behavioral groups. The gate values are mostly ranging between 0.57 and 0.70. This suggests

 doi:10.20944/preprints202604.1252.v1

that the model captures information from diverse sources rather than focusing on a single dominant modality.

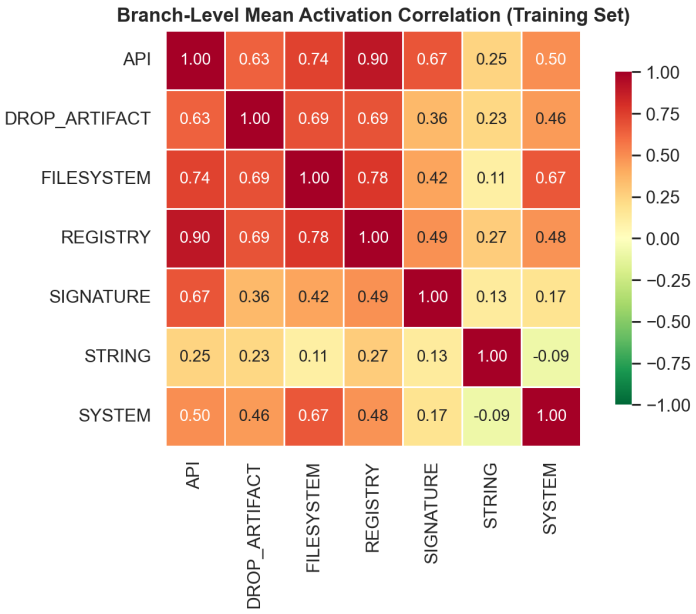


Figure 11. Branch-level correlation of feature groups based on mean activations. Strong correlations are observed among core behavioral features (API, filesystem, registry), while string-based features remain weakly correlated, indicating complementary information. (Train set).

On the other hand, despite the absence of extreme sparsity, relative differences between groups were observable. From Figure 13 groups related to signatures, API activity, and string-based features consistently received higher gate values. Whereas groups such as environment information and certain filesystem or registry discovery features received relatively lower values. The differences shown in Figure 13 confirm that the HSNN model assigns different weights to different behavioral categories. From a domain perspective, the observed importance patterns align with commonly reported behavioral indicators of ransomware, such as API activity, string artifacts, and signature-based features. This suggests that the HSNN model captures structured patterns that are consistent with the predefined behavioral taxonomy, rather than indicating causal relationships. Hence, the HSNN learns a representation in which the weighting of behavioral modalities reflects structured patterns consistent with the predefined taxonomy, while remaining data driven.

Table 11 summarizes the variation of gate values across seeds. The results show that standard deviations remain small relative to the mean gate values (typically ≈0.02–0.03), indicating that the learned importance patterns are moderately consistent across runs. While some variation is observed, no group exhibits high variability, suggesting that the hierarchical gating mechanism produces stable yet flexible representations across different initializations. In the detailed per feature results, features which are based on the signature, API activity, or are primarily composed of strings tend to receive higher importance than those that do not. Features relating to environmental information, and some of the filesystem and registry discovery features typically receive lower importance. These characteristics are also reflected in the group level statistics summarized in Table 11 and in Figure 12 and Figure 13. Figure 13 shows the ranking of the behavioral groups by their mean gate values for each seed.

Table 11. HSNN Gate Stability Across Seeds.

Group	Mean	Std (σ)	Stability
api_other	0.654	~0.03	Stable

 doi:10.20944/preprints202604.1252.v1

string_misc	0.648	~0.02	Stable
signature_other	0.654	~0.02	Stable
signature_misc	0.644	~0.03	Stable
drop_misc	0.646	~0.02	Stable
filesystem_activity	0.596	~0.02	Stable
registry_activity	0.612	~0.03	Stable
environment_info	0.584	~0.01	Highly stable

Stability labels are descriptive summaries based on relative standard deviation across seeds and are not derived from a formal threshold-based criterion.

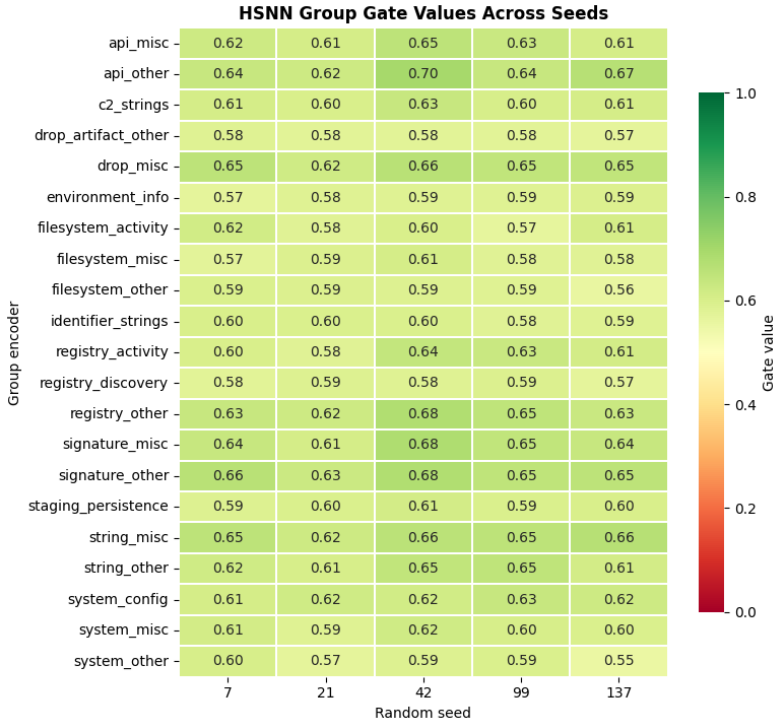


Figure 12. HSNN group gate values across random seeds. Values are stable, meaning that importance scores are consistently assigning importance to different sets of behavioral features.

Figure 14 presents the learned gate values of HSNN at both the branch and group levels. We observe that features related to API, strings, and signatures always have high gate values (>= 0.7), while features related to filesystem, system, and drop-artifact have low importance. For group level, we observe that most influential groups are: signature_misc and string_misc. Interestingly, the model does not solely rely on one feature group to learn to detect malicious packages. Instead, it uses several behavioral modalities to effectively detect package corruption. Importantly, we see that all the learned gate values are higher than 0.5, which means that we employ soft prioritization of features as opposed to hard feature sparsification. The method achieves the goal of learning feature importance weights (importance weights) across different image, spatial, and temporal modalities by focusing on relative importance weights rather than suppressing entire feature groups to zero. The importance weights learned by the network are sparse and, albeit not exactly selecting a subset of features according to a given structured prior over the gating weights, have an organized structure that is consistent with selective feature importance and follows the predefined behavioral taxonomy and hierarchy of feature groups.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 17 April 2026 doi:10.20944/preprints202604.1252.v1

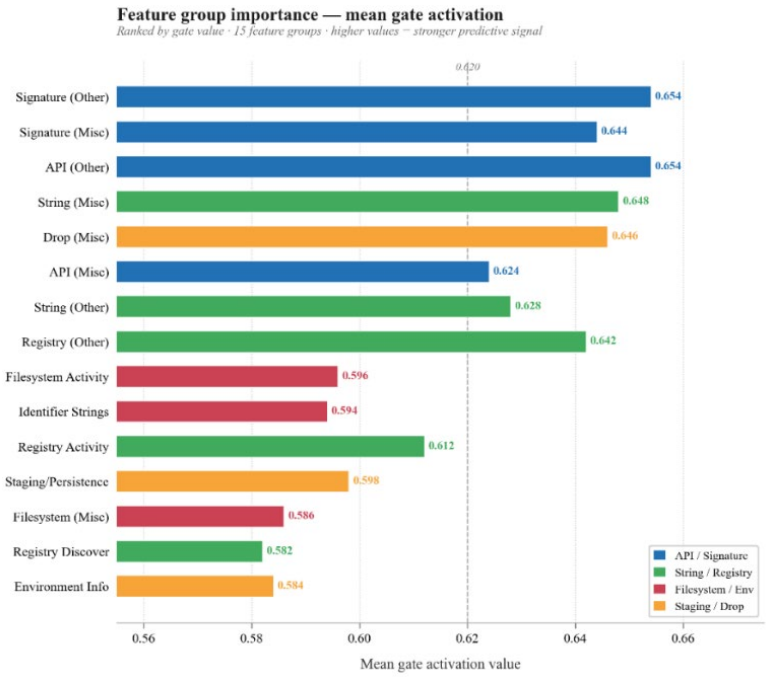


Figure 13. Mean HSNN group gate activation values sorted by how important they are for each group of behavioral features. Higher values mean that the model's predictions are more accurate.

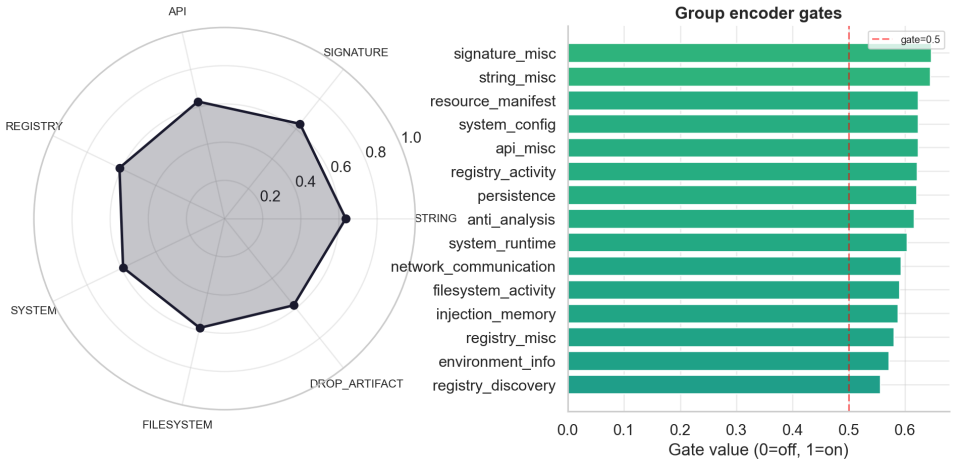


Figure 14. Learned gate values of HSNN at branch and group levels under the full feature setting. Higher values show that something is more important. The model keeps contributions from all behavioral groups, but it gives more weight to features related to API, strings, and signatures.

Figure 15 shows the evolution of the branch gate values during training for HSNN. The plots are from the last training phase where the best combination of hyperparameters is used. Our neural models are trained for 30 epochs. The branch gate values of the feature groups continuously increase in the first few epochs and plateau in the following epochs. However, the relative order of their values never changes. The string, API, and signature-related feature groups always have higher values than the other branches. Interestingly, the same ranking pattern is preserved as well for the non-sparse case, thus the relative importance values remain robust against the introduced sparsity.

 doi:10.20944/preprints202604.1252.v1

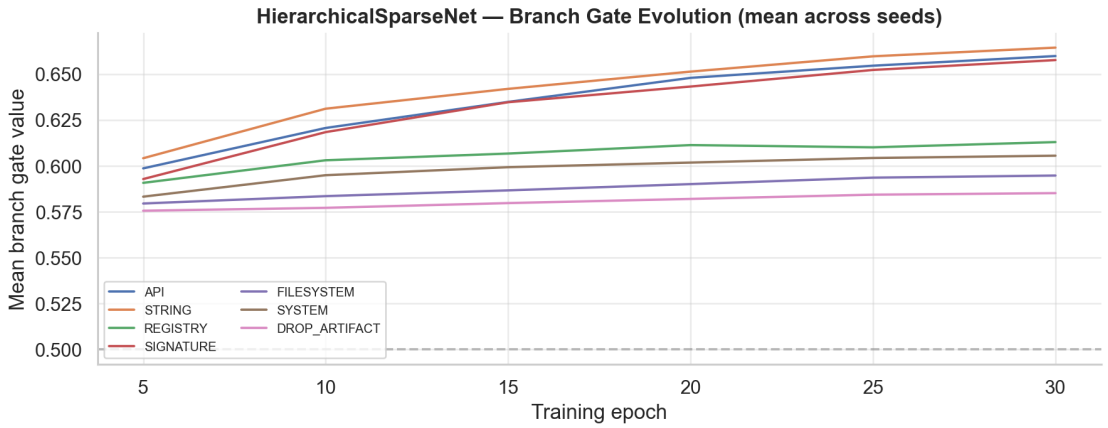


Figure 15. Evolution of branch gate values across training epochs for HierarchicalSparseNet (HSNN). During the 30-epoch training phase, the gating structure maintains constant feature group importance and stabilizes early.

For the HSNN we developed an associated gating mechanism that enables structured and interpretable prioritization of behavioral feature groups and provides insight into the contributions of individual modalities. Currently we prioritize feature groups by importance, but we do not sparsify completely: in other words, all groups of features are still above threshold, and thus we sacrifice some representation stability for this one aspect of interpretability.

3.4. RQ4: Model Reliability, Calibration, & Stability

3.4.1. Calibration of Model Predictions

The calibration of the models is evaluated using the Expected Calibration Error (ECE) and the Brier Score, which are averaged over all protocols for all features under the full feature setting and reported in Table 12. Calibration of the models varies. The HSNN has the lowest average ECE (0.0108) and Brier score (0.0122) in this scenario, which indicates that the model's confidence is well aligned with the true classification. The calibration of the HSNN model without sparsity (HSNN_NoSparsity) is almost the same as that of the HSNN model, suggesting that the hierarchical structure plays a more prominent role in calibration than sparsity alone. The calibration of the LogReg model is also strong with an average ECE of 0.0115 and a Brier score of 0.0125 and high classification performance. The calibration of the XGBoost and ElasticNetLogReg models is moderate. The calibration of the RandomForest model is the highest with an average ECE of 0.0411 and a Brier score of 0.0219. The calibration of the FlatMLP model with the highest average F1-macro score of 0.9860 has an average ECE of 0.0164 which is lower than that of the LogReg but higher than that of the HSNN.

Table 12. Performance Comparison of HSNN and Baseline Models.

Model	ECE (Avg) ↓	Brier Score (Avg) ↓	F1-Macro (Avg) ↑	ROC-AUC (Avg) ↑
HSNN	0.0108	0.0122	0.9839	0.9961
HSNN_NoSparsity	0.0108	0.0122	0.9838	0.9961
LogReg	0.0115	0.0125	0.9835	0.9973
XGBoost	0.0131	0.0176	0.9776	0.9954
ElasticNetLogReg	0.0147	0.0149	0.9806	0.9966
FlatMLP	0.0164	0.0142	0.9860	0.9949
BranchOnlyNet	0.0168	0.0153	0.9817	0.9947
SparseLinearSVM	0.0198	0.0143	0.9805	0.9979

 doi:10.20944/preprints202604.1252.v1

RandomForest	0.0411	0.0219	0.9775	0.9947

While the FlatMLP achieved the highest F1-Macro score (0.9860) and the SparseLinearSVM led in ROC-AUC (0.9979), the HSNN stands out as one of the most reliable models overall. This is primarily due to its strong calibration; the HSNN produced the lowest expected calibration error at 0.0108. In practical terms, this means that while other models may achieve slightly higher accuracy or ranking performance, the HSNN's predicted probabilities are more well-calibrated and better reflect the true likelihood of outcomes.

3.4.2. Models Stability Across Random Seeds

The stability of the developed models was assessed. Models' performance was assessed by calculating the mean, standard deviation, minimum, maximum and coefficient of variation (CV) of the performance for all performance metrics across all random seeds (Table 13).

Table 13. Stability Statistics of Model Performance Across Random Seeds (F1-Macro).

Model	Mean F1	Std. Dev.	Min F1	Max F1	CoV
FlatMLP	0.9817	0.0034	0.9774	0.9866	0.0034
HSNN_NoSparsity	0.9803	0.0020	0.9774	0.9825	0.0020
BranchOnlyNet	0.9797	0.0030	0.9764	0.9846	0.0031
HSNN	0.9795	0.0019	0.9774	0.9815	0.0020

Table 13 displays the standard deviations and coefficients of variation (CoV) for each model, as well as the range of performance across seeds. The low variability and small standard deviation result in small coefficients of variation indicating stable reproduction performance across randomizations. In summary, HSNN exhibits the lowest variability among the evaluated models. The range of performance values, measured by the difference between maximum and minimum F1 scores, is also smaller for HSNN than for the other models. Figure 16 presents the distribution plots of performance in terms of F1-macro, balanced accuracy, and ROC-AUC. The distributions are concentrated for all models, with HSNN showing a slightly more concentrated distribution compared to FlatMLP.

Figure 16 presents the distribution of performance metrics across five random seeds, further illustrating the stability patterns observed in Table 13. While Table 13 focuses on performance stability, we further examine the consistency of feature selection across seeds in Table 14.

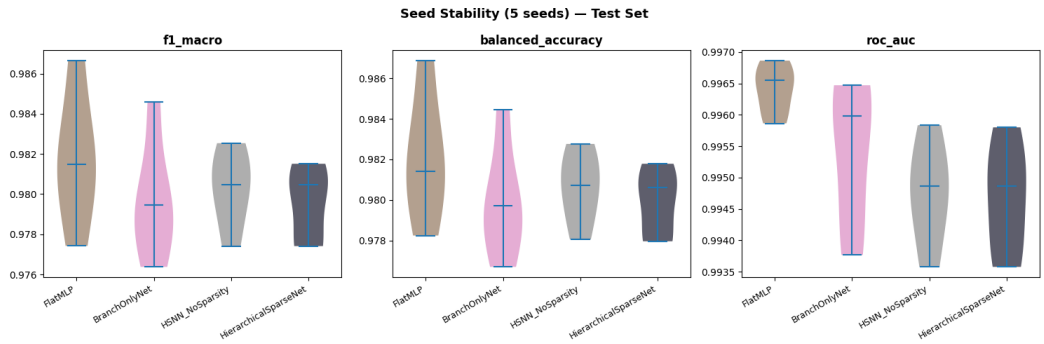


Figure 16. Distribution of model performance across five random seeds for F1-macro, balanced accuracy, and ROC-AUC. All models exhibit tightly clustered distributions, with HSNN showing marginally lower variability across runs.

Table 14 highlights the stability of the model's feature selection; specifically, it shows the Jaccard similarity for the top-k feature groups across multiple HSNN training seeds. The average and

 doi:10.20944/preprints202604.1252.v1

standard deviation over the five seeds (7, 21, 42, 99, 137) are provided. The Jaccard similarity measures the overlap between the sets of selected feature groups across seeds. As shown in Table 14, the overlaps are moderate, indicating that the same sets of feature groups are selected with moderate consistency across runs, while allowing variation in the specific subsets of selected groups. This suggests that the HSNN maintains moderately stable feature selection patterns across seeds, while permitting limited variability in the learned behavioral representations.

Table 14. Summary statistics and pairwise Jaccard similarity of top-k feature group selections across HSNN runs with different random seeds. Summary statistics for this set of runs have a mean overlap of 0.531; however, the pairwise overlap is not uniform across runs.

(a) Summary Statistics		**(b) Pairwise Jaccard Similarity Matrix (Top-k Feature Groups)**					
Statistic	Value	Seeds	7	21	42	99	137
Mean Jaccard	0.531	7	—	0.429	0.667	0.429	0.429
Std. Dev.	0.187	21	0.429	—	0.667	0.667	1.000
Min	0.429	42	0.667	0.667	—	0.429	0.667
Max	1.000	99	0.429	0.667	0.429	—	0.667
		137	0.429	1.000	0.667	0.667	—

4. Discussion

We compare our architecture, HSNN, against flat neural networks as well as against several strong baselines. All networks achieve high peak accuracy, but they have quite different efficiency, calibration, and representation stability.

RQ1: Predictive Power and Operational Efficiency

In terms of predictive power and operational efficiency, the best performer was the FlatMLP, achieving the highest accuracy and efficiency with an average macro-F1 score of 0.9860. The scores of the HSNN (0.9839) were only a few percentage points behind, and this equivalent performance is achieved with a 42% reduction in parameters. Therefore, the HSNN can be used as an alternative to a much larger FlatMLP for practical applications.

RQ2: Robustness and Generalization across Deployment Scenarios

In terms of how these models behave under distribution shift, we observe that model rankings are not static; they change when moving from controlled i.i.d. settings to more challenging scenarios such as open-set evaluation. While the FlatMLP led in standard evaluations, the HSNN surprisingly dominated the Open-Set (P4) protocol, outperforming the baseline (0.9930 vs. 0.9913). While performance degradation is observed under temporal and open-set protocols, this study does not include a fine-grained subset-level error analysis. The results suggest that distribution shift may influence misclassification patterns; however, identifying specific failure sources is left for future work. These findings suggest that the HSNN shows comparatively stronger relative ranking under the family-disjoint and open-set protocols evaluated in this paper, which are good representative of real-world deployment scenarios.

RQ3: Feature Architecture, Redundancy, and Interpretability

Our investigation into the model's "inner logic" showed that string-based artifacts remain the most dominant predictive signal, representing 46.2% of the feature space. Despite the network being heavily gated, it still relied on these strings; in fact, performance dropped most significantly when the STRING modality was removed. However, the

HSNN is not simply over-fitting to noise. When we performed feature decorrelation to remove redundant information, the macro-F1 scores remained invariant. In addition to good performance on the ransomware detection task, we also observe high taxonomic coherence in the model. That is, the activity of the learned gate corresponds well with the feature taxonomy and importance of the

 doi:10.20944/preprints202604.1252.v1

modalities. This indicates that the HSNN is learning a decision-making strategy that is structured and interpretable and corresponds to meaningful groupings of features in the domain.

RQ4: Reliability, Calibration, and Representational Stability

Beyond raw accuracy, we looked at how much we can actually trust these models. While models like the SparseLinearSVM achieved high ROC-AUC (0.9979), the HSNN proved to be the most reliable overall. Looking at the expected calibration error for the probability output of the models, the best performing model for this was the HSNN at 0.0108. This actually performed better than the highest accuracy model (FlatMLP) which had a brier score of 0.0122. We also found that the HSNN exhibits consistent behavior across different random seeds. It demonstrated moderately consistent representational patterns across random seeds, with a mean Jaccard similarity of 0.531 for its top-k gates. This indicates that the HSNN captures stable structural patterns in the data, rather than relying on seed-specific variations.

4.1. Summary of Findings

Table 15 summarizes the key findings aligned with each research question, emphasizing performance, robustness under distribution shift, feature-level behavior, and calibration reliability. Whereas individual metrics provide some information, our results suggest that no single metric is sufficient to characterize model quality for behavioral ransomware detection. Although the FlatMLP achieved the highest average macro-F1 score with a small margin, the margin was not consistent across protocols. Moreover, HSNN achieved detection performance that was comparable, and in some protocols higher, with better calibration, fewer parameters, and relative performance improvements over the baselines that were more pronounced in the family-disjoint and open-set settings. A closer examination of the features and gates of the HSNN model reveals that the string-based artifacts were as significant as before, that the decorrelation had almost no effect on model performance, and that the gate importances were stable and domain-consistent across the different behavioral modalities.

Table 15. Summary Research Questions Mapping with Findings.

RQ	Research Focus	Primary Metrics	Core Finding
RQ1	Performance & Efficiency	Macro-F1, Parameter Count	HSNN achieved performance comparable to FlatMLP while reducing model size by 42%, indicating an improved efficiency–performance trade-off.
RQ2	Robustness & Generalization	Protocol-wise Rank Comparison, Open-Set F1	HSNN demonstrates protocol-dependent ranking behavior and strong performance under the Open-Set (P4) protocol (0.9930), suggesting robustness under distribution shift.
RQ3	Feature Architecture & Interpretability	ΔF1 (Ablation), Decorrelation F1	STRING features remain among the most influential predictive signals. HSNN maintains performance under feature decorrelation and exhibits structured feature utilization aligned with the predefined taxonomy.
RQ4	Reliability & Stability	ECE, Brier Score, Gate Jaccard Similarity	HSNN achieves the lowest calibration error (ECE: 0.0108) and demonstrates moderately consistent internal representations across random seeds.

 doi:10.20944/preprints202604.1252.v1

5. Conclusions and Future Works

5.1. Conclusions

Most current studies on the detection efficacy of different behavioral ransomware (BRW) detection algorithms are based on certain testbeds. This work investigates a more comprehensive and practical approach to BRW detection. In this paper, we propose an integrated experimental framework to test ransomware detection models under various experimental conditions, such as evaluation approaches, feature sets, and reliability, and then design a Hierarchical Sparse Neural Network (HSNN) based on both group-level and branch-level features to capture structured behavior of ransomware.

We show that, although the baseline FlatMLP attains high average detection performance, it captures very little of the actual model behavior. While the best detector under standard (stratified) settings only achieved detection performance comparable to that of the HSNN network, the latter also achieved better calibration, stability, and robustness, especially under more challenging scenarios such as family-disjoint and open-set settings. Interestingly, the model rankings drastically change across evaluation settings. While some models might be ranked highest for standard stratified settings, they would have the poorest performance in temporal, cross-family and open-set settings. This highlights the importance of evaluating detectors on settings more realistic than simply a single stratified split.

Although string-based features are not the most powerful tools for modern ransomware detection, they still have some utility. These features are very unpredictable and have a strong potential for overfitting and bias. Although there is significant correlation between many features, removing the most redundant ones results in a small decrease in accuracy. Importantly, the importance patterns learned by the HSNN model correspond to typical behavioral indicators of ransomware and are consistent across repeated experiments. The HSNN therefore provides a good balance between traditional machine learning methods and flat neural networks, in terms of both accuracy and calibration/stability/robustness across protocols. This paper argues that we need tree evaluation methods that go further than single split point performance.

While this study conducted many tests (on different protocols, and on different hyper-parameters), a further experiment could still be tested in future. First, all the experiments were run on the MLRan dataset, running more experiments on other datasets would help generalize results. Secondly, while P3 maintains the separation of ransomware families, the benign samples are split across groups leading to mild overlap in data distributions. Thirdly, while the open-set detection used is straightforward (maximum softmax probability), it may not be the current state of the art for novelty detection. Finally, while the trends across protocols are consistent, it would be good to include tests of significance where appropriate. We observe that the gate stability does not directly translate to substantial overlap in feature subspaces selected by individual gates. Additionally, we measure efficiency primarily in terms of the number of parameters and latency required to compute the output of individual gates (as opposed to a full system level analysis). We discuss these limitations in the future work section.

We discuss how the current single-split accuracy-oriented benchmarking methods might not be sufficient to evaluate ransomware behavioral detection approaches. Instead, we propose the use of more elaborate benchmarking methods focusing on protocols and robustness.

5.2. Future Work

To take this study to the next level, there are several paths we can explore. Firstly, it would be really helpful to test our model with different datasets and see how it performs in various scenarios. This will give us a better idea of its strengths and weaknesses. We should also try to make our evaluation process more realistic, so we can get a clearer picture of how ransomware evolves over time and how our model's performance changes as a result. Another area for improvement is in identifying unknown types of ransomware. Currently, we're using a basic method, but we could get

 doi:10.20944/preprints202604.1252.v1

better results by using more advanced techniques to measure uncertainty. Additionally, we need to make sure our model can withstand attempts to manipulate it, especially when it comes to important features like string and API artifacts. Lastly, we could enhance our architecture by incorporating more flexible and adaptive ways of learning, as well as getting a better sense of how it will perform in real-world situations. This could involve combining different approaches or taking a closer look at how it works on a system level. By exploring these avenues, we can make our model more effective, easier to understand, and more practical to use.

Author Contributions: I.K.N: Conceptualization, methodology, validation, formal analysis, investigation, writing—original draft preparation, Visualization. The author read and agreed to the published version of the manuscript.

Data availability: All datasets used in this study are publicly available on Hugging Face. Source codes will be made available on GitHub.

Declaration of competing interest: The authors state that they have no known financial or personal conflicts of interest that could have influenced the work presented in this study.

Funding Declaration: The authors declare that no external funding was received for this research.

References

1. Oz, H.; Aris, A.; Levi, A.; Uluagac, A.S. A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. *ACM Comput. Surv.* **2022**, *54*, 1–37, doi:10.1145/3514229.
2. Hirano, M.; Kobayashi, R. RanSMAP: Open Dataset of Ransomware Storage and Memory Access Patterns for Creating Deep Learning Based Ransomware Detectors. *Comput. Secur.* **2025**, *150*, 104202, doi:10.1016/j.cose.2024.104202.
3. Wiles, A.; Colombo, F.; Mascorro, R. Ransomware Detection Using Network Traffic Analysis and Generative Adversarial Networks. *Authorea preprint* **2024**, doi:10.22541/au.172659907.77469627/v1.
4. Sibtain, M.; Hussain, M.; Riaz, Q.; Qadir, S.; Riaz, N.; Jung, K.-H. Lightweight and Robust Android Ransomware Detection Using Behavioral Analysis and Feature Reduction. *Computers, Materials & Continua* **2025**, *84*, 5177–5199, doi:10.32604/cmc.2025.066198.
5. Urooj, U.; Al-Rimy, B.A.S.; Zainal, A.; Ghaleb, F.A.; Rassam, M.A. Ransomware Detection Using the Dynamic Analysis and Machine Learning: A Survey and Research Directions. *Applied Sciences (Switzerland)* **2022**, *12*, doi:10.3390/app12010172.
6. Alzakari, S.A.; Aljebreen, M.; Ahmad, N.; Alhashmi, A.A.; Alahmari, S.; Alrusaini, O.; Al-Sharafi, A.M.; Almukadi, W.S. An Intelligent Ransomware Based Cyberthreat Detection Model Using Multi Head Attention-Based Recurrent Neural Networks with Optimization Algorithm in IoT Environment. *Sci. Rep.* **2025**, *15*, 8259, doi:10.1038/s41598-025-92711-4.
7. Federal Bureau of Investigation, I.C.C.C. (IC3) *2023 Internet Crime Report*; Washington, DC, USA;
8. Anna, R. FBI's Internet Crime Report 2024 Records $16.6 Billion in Cybercrime Losses amid Rising Ransomware Threats Available online: https://industrialcyber.co/reports/fbis-internet-crime-report-2024-records-16-6-billion-in-cybercrime-losses-amid-rising-ransomware-threats/#:~:text=The Federal Bureau of Investigation,$800 million in ransom payments.
9. Garter, L.; Johnson, C.; Brown, A.; Miller, W.; Davis, M.; Martin, D. A Novel Approach of Ransomware Detection Using Dynamic Behavior Modeling and Network Pattern Profiling 2024.
10. Verizon Business 2025 Data Breach Investigations Report Available online: https://its.ny.gov/system/files/documents/2025/06/maguire-verizon.pdf (accessed on 23 March 2026).
11. IBM Security *Cost of a Data Breach Report 2024*; 2024;
12. Sophos *The State of Ransomware 2025*; 2025;
13. Aurangzeb, S.; Anwar, H.; Naeem, M.A.; Aleem, M. BigRC-EML: Big-Data Based Ransomware Classification Using Ensemble Machine Learning. *Cluster Comput.* **2022**, *25*, 3405–3422, doi:10.1007/s10586-022-03569-4.

 doi:10.20944/preprints202604.1252.v1

14. Ring, M.; Wunderlich, S.; Scheuring, D.; Landes, D.; Hotho, A. A Survey of Network-Based Intrusion Detection Data Sets. *Comput. Secur.* **2019**, *86*, 147–167, doi:10.1016/j.cose.2019.06.005.
15. Kharraz, A.; Robertson, W.; Balzarotti, D.; Bilge, L.; Kirda, E. Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks. In; 2015; pp. 3–24.
16. Brewer, R. Ransomware Attacks: Detection, Prevention and Cure. *Network Security* **2016**, *2016*, 5–9, doi:10.1016/S1353-4858(16)30086-1.
17. Paquet-Clouston, M.; Haslhofer, B.; Dupont, B. Ransomware Payments in the Bitcoin Ecosystem. *J. Cybersecur.* **2019**, *5*, doi:10.1093/cybsec/tyz003.
18. Almuflih, A.S. Two-Tier Heuristic Search for Ransomware-as-a-Service Based Cyberattack Défense Analysis Using Explainable Bayesian Deep Learning Model. *Sci. Rep.* **2026**, *16*, 437, doi:10.1038/s41598-025-96083-7.
19. Kritika, Er. A Comprehensive Literature Review on Ransomware Detection Using Deep Learning. *Cyber Security and Applications* **2025**, *3*, 100078, doi:10.1016/j.csa.2024.100078.
20. Cybersecurity and Infrastructure Security Agency (CISA) Stop Ransomware Guide Available online: https://www.cisa.gov/stopransomware (accessed on 23 March 2026).
21. BOUTEMEUR, J.; LELLA, I.; BAKATSIS, I.; CHATZICHRISTOS, G.; FOLEY, K.; LESKINEN, J.; OTCENASEK, J.; ZIOLEK, D.; ENISA *ENISA Threat Landscape 2025*; 2025;
22. Fisher, D.; Evans, D.; Clark, S.; Lewis, J.; Walker, A.; Hall, N.; Young, V. Hierarchical Behavioral Pattern Analysis for Real-Time Ransomware Detection 2024.
23. Davidian, M.; Kiperberg, M.; Vanetik, N. Early Ransomware Detection with Deep Learning Models. *Future Internet* **2024**, *16*, 291, doi:10.3390/fi16080291.
24. Nti, I.K.; Nyarko-Boateng, O. SHAP-Guided Feature Refinement for Robust and Interpretable Malware Detection in Memory Forensics. *Research Square preprint* **2025**, doi:10.21203/rs.3.rs-7871553/v1.
25. Hossain, Md.A.; Saif, S.; Islam, Md.S. A Novel Federated Learning Approach for IoT Botnet Intrusion Detection Using SHAP-Based Knowledge Distillation. *Complex & Intelligent Systems* **2025**, *11*, 422, doi:10.1007/s40747-025-02001-9.
26. Qian, L.; Cong, L. Channel Features and API Frequency-Based Transformer Model for Malware Identification. *Sensors* **2024**, *24*, 580, doi:10.3390/s24020580.
27. Qasim, H.; Lu, Y. Systematic Evaluation Framework for ML and DL-Based Ransomware Detection 2026.
28. Bruschi, D.; De Corato, M.; Ferrara, A.; Salini, S. Ransomware Detection Using Sample Entropy and Graphical Models: A Methodology for Explainable Artificial Intelligence (XAI) in Cybersecurity. *Appl. Stoch. Models Bus. Ind.* **2025**, *41*, doi:10.1002/asmb.70061.
29. Moujoud, L.; Ayache, M.; Belmekki, A. A State-of-the-Art Survey on Ransomware Detection Using Machine Learning and Deep Learning. In; 2023; pp. 183–200.
30. Alraizza, A.; Algarni, A. Ransomware Detection Using Machine Learning: A Survey. *Big Data and Cognitive Computing* **2023**, *7*, 143, doi:10.3390/bdcc7030143.
31. Zhang, K.; Wang, Y.; Bhatti, U.A.; Zhou, Y.; Jin, M. Enhanced Ransomware Attacks Detection Using Feature Selection, Sensitivity Analysis, and Optimized Hybrid Model. *J. Big Data* **2025**, *12*, doi:10.1186/s40537-025-01289-1.
32. Tugba Yazan, D. Building Trust in Malware Detection: Interpretable Multi-Label Model with Visual Feature Attribution Name, Birkbeck, University of London: London, 2025.
33. Onwuegbuche, F.C.; Adelodun, S.O.; Jurcut, A.D.; Pasquale, L. MLRan: A Behavioural Dataset for Ransomware Analysis and Detection. *Journal of Network and Computer Applications* **2026**, *250*, 104475, doi:10.1016/j.jnca.2026.104475.
34. Kapoor, S.; Narayanan, A. Leakage and the Reproducibility Crisis in Machine-Learning-Based Science. *Patterns* **2023**, *4*, 100804, doi:10.1016/j.patter.2023.100804.
35. Semmelrock, H.; Ross-Hellauer, T.; Kopeinik, S.; Theiler, D.; Haberl, A.; Thalmann, S.; Kowald, D. Reproducibility in Machine-learning-based Research: Overview, Barriers, and Drivers. *AI Mag.* **2025**, *46*, doi:10.1002/aaai.70002.

 doi:10.20944/preprints202604.1252.v1

36. Pendlebury, F.; Pierazzi, F.; Jordaney, R.; Kinder, J.; Cavallaro, L.; London, C.; Holloway, R. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In Proceedings of the 28th USENIX Security Symposium; USENIX Association: Santa Clara, CA, USA, August 16 2019; pp. 729–746.
37. Arp, D.; Pendlebury, F.; London, C.; Holloway, R.; Pierazzi, F.; Quiring, E.; Warnecke, A.; Wressnegger, C.; Cavallaro, L.; Rieck, K. Dos and Don'ts of Machine Learning in Computer Security. In Proceedings of the 31st USENIX Security Symposium; USENIX Association: Boston, MA, USA, August 12 2022; pp. 3971–3988.
38. Kan, Z.; McFadden, S.; Arp, D.; Pendlebury, F.; Jordaney, R.; Kinder, J.; Pierazzi, F.; Cavallaro, L. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time (Extended Version). **2025**.
39. Hendrycks, D.; Gimpel, K. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. **2018**.
40. Anderson, B.; McGrew, D. Machine Learning for Encrypted Malware Traffic Classification. In Proceedings of the Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; ACM: New York, NY, USA, August 13 2017; Vol. Part F129685, pp. 1723–1732.
41. Guyon, I.; Elisseeff, A.; Kaelbling, L.P. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research* **2003**, *3*, 1157–1182, doi:10.1162/153244303322753616.
42. Dormann, C.F.; Elith, J.; Bacher, S.; Buchmann, C.; Carl, G.; Carré, G.; Marquéz, J.R.G.; Gruber, B.; Lafourcade, B.; Leitão, P.J.; et al. Collinearity: A Review of Methods to Deal with It and a Simulation Study Evaluating Their Performance. *Ecography* **2013**, *36*, 27–46, doi:10.1111/j.1600-0587.2012.07348.x.
43. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32, doi:10.1023/A:1010933404324.
44. Chen, T.; Guestrin, C. XGBoost. In Proceedings of the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; ACM: New York, NY, USA, August 13 2016; pp. 785–794.
45. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. **2019**.
46. Loshchilov, I.; Hutter, F. Decoupled Weight Decay Regularization. **2019**.
47. Prechelt, L. Early Stopping - But When? In; 1998; pp. 55–69.
48. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna. In Proceedings of the Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; ACM: New York, NY, USA, July 25 2019; pp. 2623–2631.
49. Fabian, P.; Ga¨el, V.; Alexandre, G.; Michel, V.; Bertrand, T.; Olivier, G.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Vincent, D.; et al. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.
50. Guo, C.; Pleiss, G.; Sun, Y.; Weinberger, K.Q. On Calibration of Modern Neural Networks. In Proceedings of the 34th International Conference on Machine Learning; 2017; pp. 1321–1330.