Beyond IDS Signatures:

AI-Powered Ransomware Detection Through Memory and Storage Forensics

Cybersecurity and AI Systems Engineering

Prepared by

Oussama Ben Zian
Monssef Tlidi

Supervised by

Professor Saiida Lazaar

April 2025

— ABSTRACT —

In this research, we set out to address one of the most pressing challenges in cybersecurity today: the detection of modern ransomware that evolves too quickly for traditional, signature-based Intrusion Detection Systems (IDS) to keep up. While tools like Snort and Suricata remain essential for identifying known threats, they often fail when faced with novel or obfuscated variants that bypass rule-based detection.

To confront this, we built and evaluated a series of lightweight, AI-driven behavioral models—including Autoencoders, Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) architectures—trained on the RanSMAP dataset, a rich telemetry source of storage and memory-level activity collected at fine time intervals. Our models were designed for CPU-only environments, ensuring practical deployability alongside existing IDS engines.

Early experiments under a random-split setting yielded near-perfect results, with LSTM-based models achieving over 99% accuracy and F1-scores—highlighting the ability of neural models to recognize known ransomware behaviors. However, the real challenge emerged when we introduced the more realistic *family-aware split*, where test ransomware families remained completely unseen during training. This split better simulates real-world zero-day detection scenarios and revealed the generalization limitations of simpler models like CNNs or Autoencoders.

Through extensive experimentation—covering multiple architectures, hyperparameters, and data preparation strategies—we discovered that combining local pattern extraction (via CNN) with sequence modeling (via LSTM) dramatically enhanced detection robustness. Our best-performing architecture, a dual-branch CNN–LSTM with bidirectional LSTM layers and parallel convolutional streams, achieved an F1-score of 92.24% and recall of 91.02% on unseen ransomware families—significantly outperforming all baseline models.

Though not yet implemented, we also explored the potential of integrating wavelet transforms into our pipeline. Prior research suggests that wavelets can reduce noise and highlight behavioral anomalies, which may enhance both training stability and detection accuracy in future iterations.

Beyond practical performance, this project also offered academic insights into the behavior of sequential ransomware telemetry data. We learned the importance of family-aware splitting, the risk of over-relying on random splits, and which preprocessing steps—like scaling, sequence generation, and latent-space structuring—most impacted model reliability. These lessons are valuable not only for our project but for the broader field of AI-enhanced malware detection.

This work ultimately reinforces the value of hybrid detection approaches: models that do not replace traditional IDS, but rather complement them—adding an intelligent behavioral layer capable of catching what static rules often miss.

Contents

1 Introduction

Ransomware continues to inflict devastating financial and operational impacts on organizations worldwide. Despite the ubiquity of signature-based IDS tools such as Snort and Suricata, these systems struggle to detect zero-day and polymorphic variants that evade known signatures. Meanwhile, the surge of AI applications in cybersecurity promises adaptive, behavior-based detection capable of uncovering previously unseen attacks.

In this work, we explore whether lightweight AI models—suitable for CPU-only environments—can effectively detect ransomware based on low-level behavioral patterns, and complement existing rule-based engines. We focus on three distinct neural architectures:

- **Autoencoder** (unsupervised anomaly detection) learns baseline behavioral profiles and flags deviations.
- **Convolutional Neural Network (CNN)** treats structured temporal features as spatial patterns for supervised binary classification.
- **Long Short-Term Memory (LSTM)** models the sequential evolution of memory and storage access behaviors.

To further bolster detection and reduce CPU overhead, we introduce **Discrete Wavelet Transforms (DWT)** to preprocess time-series features, attenuating noise and lowering dimensionality. Our evaluation uses the publicly available **RanSMAP dataset**, which captures ransomware and benign application behavior via storage and memory access patterns collected through a thin hypervisor. We measure each model's accuracy, recall, F1-score, false-positive rate, and inference latency, both with and without wavelet-enhanced features.

By the end of this study, we aim to demonstrate a practical "shadow hybrid" architecture (Figure 1), where a traditional IDS operates in real time while our ML layer runs in parallel, merging alerts to improve detection coverage without altering legacy systems.

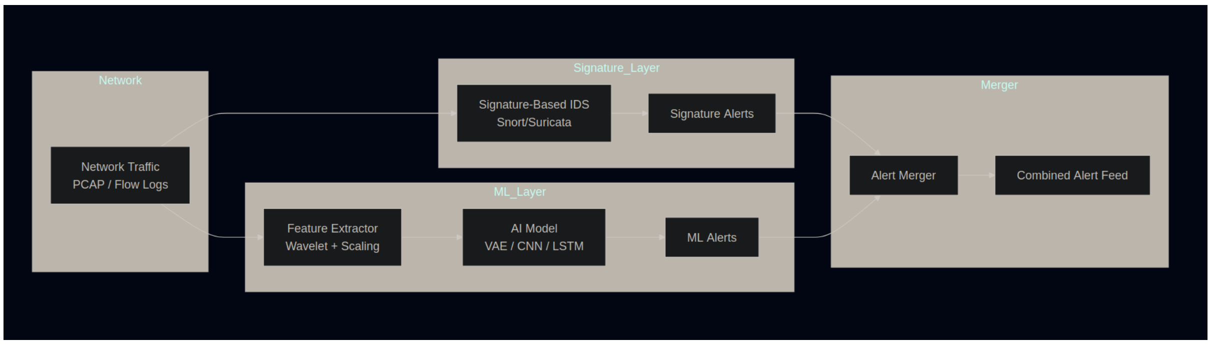

Network
Network Traffic PCAP / Flow Logs
Signature_Layer
Signature-Based IDS Snort/Suricata
Signature Alerts
ML_Layer
Feature Extractor Wavelet + Scaling
AI Model VAE / CNN / LSTM
ML Alerts
Merger
Alert Merger
Combined Alert Feed

Figure 1: Hybrid IDS architecture: rule-based and AI-driven layers in parallel (placeholder)

2 Background & Related Work

2.1 Traditional Signature-Based Intrusion Detection Systems

Signature-based IDS, such as Snort and Suricata, rely on predefined rules and patterns to identify malicious traffic. These systems are highly efficient at matching known exploits and vulnerabilities, often operating at line-rate in production environments. However, their dependence on up-to-date signature databases creates a blind spot for zero-day attacks and polymorphic malware that alter their payloads or communication behaviors to evade detection [**?**, **?**]. Moreover, managing and tuning thousands of rules can lead to high false-positive rates and administrative overhead.

2.2 Machine Learning-Based IDS

Machine learning (ML) techniques offer behavior-driven detection by learning patterns from data rather than relying on fixed signatures. For non-expert readers, think of ML models as statistical "pattern detectives" that digest examples of normal and malicious traffic, then decide whether new observations fit the learned patterns. We introduce three complementary architectures below:

2.2.1 Autoencoder (Anomaly Detection)

An autoencoder is an unsupervised neural network comprised of two parts: an "encoder" that compresses input data into a compact representation, and a "decoder" that reconstructs the original data from this compressed form. Trained only on benign samples, the autoencoder learns to recreate familiar patterns with low error. When presented with ransomware traffic, which deviates from the training norm, the reconstruction error spikes—flagging anomalies. This makes autoencoders particularly useful for detecting novel or zero-day attacks that lack prior signatures.

2.2.2 Convolutional Neural Network (CNN)

Originally developed for image processing, CNNs apply small sliding filters across data to identify local features and patterns. For network flows, we treat each feature vector like a one-dimensional "image," where convolutional filters capture important interactions (for example, packet size combined with timing). A CNN classifier learns to map these extracted patterns to labels (benign or ransomware) through supervised training. Thanks to their shared filters, CNNs can be lightweight yet powerful at spotting subtle, distributed signals across features.

2.2.3 Long Short-Term Memory Network (LSTM)

LSTMs are a type of recurrent neural network designed to handle sequential data by maintaining an internal memory state. They excel at modeling time-dependent behavior, such as a sequence of packet inter-arrival times or a series of flow statistics over a window. In ransomware detection, LSTMs can recognize the distinct temporal signatures of an attack—such as a burst of encryption-related traffic followed by command-and-control communications—by remembering and weighing past observations when making each prediction.

2.3 Wavelet Transforms in Cybersecurity

Discrete wavelet transforms provide a time-frequency decomposition of signals, allowing denoising and dimensionality reduction without significant information loss. Prior research has shown that applying DWT to network traffic signals can improve multiclass intrusion detection accuracy by up to 3% and reduce false-positive rates [?, ?]. In our pipeline, we denoise per-flow time-series features before feeding them into ML models.

2.4 Hybrid IDS Architectures

Combining traditional and ML-based IDS in a hybrid framework leverages the strengths of both: signature engines for known threats and AI layers for novel attack detection. Architectures range from simple post-processing of alerts to deeply integrated plugins in rule engines. We adopt a "shadow hybrid" design where the two systems run in parallel and feed a merger component for consolidated alerting [?].

Architecture	**Description**	**Strengths**	**Limitations**
Signature-only	Snort/Suricata using rule-based signatures	Efficient, low-latency detection of known threats	Blind to zero-day attacks; requires constant signature updates
AI-only	Autoencoder / CNN / LSTM models for behavior-based detection	Adaptive to novel or obfuscated ransomware; can detect zero-day anomalies	Higher CPU usage; less interpretable decisions
Shadow hybrid	Parallel signature engine + ML layer with alert merging	Combines high recall on both known and unknown threats	Increased system complexity; moderate resource overhead

Table 1: Comparison of IDS architectures: signature-only, AI-only, and hybrid

3 Dataset Description

The dataset employed in this work has been specifically designed to capture detailed behavioral patterns of software execution, with the objective of distinguishing ransomware from benign programs. It was constructed through a thorough process of low-level storage and memory access monitoring, followed by preprocessing and aggregation. This resulted in a rich set of statistical features representing program behaviors across various execution scenarios.

3.1 Dataset Overview and Dimensions

The dataset comprises a total of 194,466 samples (rows), with each sample representing program behavior within a fixed-length time window. Each sample contains 26 columns, structured as 23 statistical features and 3 meta-information fields.

3.1.1 Class Distribution

- **Benign samples (label = 0)**: 101,443
- **Ransomware samples (label = 1)**: 93,023

3.1.2 Families and Sample Counts

The dataset includes samples from 12 distinct software families, representing both ransomware and benign software:

- Firefox: 20,207
- AESCrypt: 19,557
- Zip: 18,295
- LockBit: 18,093
- Conti: 17,739
- SDelete: 17,332
- Darkside: 16,771
- REvil: 15,296
- WannaCry: 14,488
- Office: 13,468
- Idle: 12,584
- Ryuk: 10,636

Among them, families such as Firefox, Idle, and Office represent benign applications, while the rest are ransomware.

3.2 Sampling Methodology

Each sample is extracted using a **fixed time window of 0.1 seconds**. During this window, storage and memory operations are monitored and aggregated into statistical representations. This approach ensures that each row reflects the behavior of the program in a short, consistent snapshot, making it ideal for machine learning models that require fixed-length input vectors.

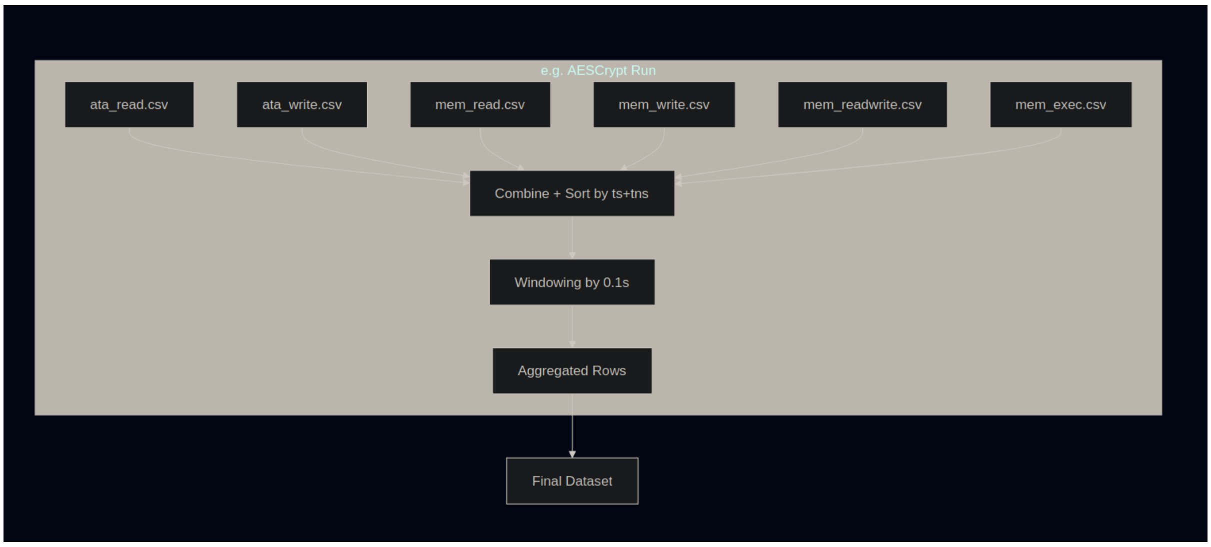


Figure 2: Dataset processing flow: from raw CSV files to aggregated rows per 0.1s window and final dataset.

3.3 Dataset Structure

Each row consists of:

- **23 statistical features**: derived from storage and memory access patterns.
- **3 meta-information fields**:
  - *family*: The program family name.
  - *window_id*: A unique identifier for the time window.
  - *label*: The binary classification label (0 for benign, 1 for ransomware).

3.3.1 Storage Access Features

These features summarize read and write behaviors to SSD storage:

- *read_throughput*: Average read throughput in bytes/second.
- *read_lba_var*: Variance of Logical Block Addresses (LBAs) during read operations.
- *write_throughput*: Average write throughput in bytes/second.
- *write_lba_var*: Variance of LBAs during write operations.
- *write_entropy*: Average Shannon entropy of write operations.

3.3.2 Memory Access Features

Memory-related features are computed based on different access types (read, write, read/write, and execute):

- *mem_read_count_4k*, *mem_read_count_2m*, *mem_read_count_mmio*: Number of read accesses categorized by page size and MMIO.
- *mem_read_gpa_var*: Variance of Guest Physical Addresses (GPA) during read operations.
- *mem_write_entropy*: Average Shannon entropy of memory write operations.
- *mem_write_count_4k*, *mem_write_count_2m*, *mem_write_count_mmio*: Number of write accesses categorized by page size and MMIO.
- *mem_write_gpa_var*: Variance of GPA during write operations.
- *mem_readwrite_entropy*: Average entropy for combined read/write operations.
- *mem_readwrite_count_4k*, *mem_readwrite_count_2m*, *mem_readwrite_count_mmio*: Counts for combined read/write accesses.
- *mem_readwrite_gpa_var*: Variance of GPA during combined read/write operations.
- *mem_exec_count_4k*, *mem_exec_count_2m*, *mem_exec_count_mmio*: Number of instruction fetches.
- *mem_exec_gpa_var*: Variance of GPA during instruction fetch operations.

3.4 Summary

In conclusion, each row in the dataset provides a detailed and structured summary of program behavior observed over a short, fixed period. The combination of storage access patterns, memory usage statistics, and entropy calculations offers a comprehensive view that is well-suited for training machine learning models to detect ransomware. While 23 features capture quantitative behavior, 3 meta fields (*family*, *window_id*, and *label*) provide context and classification information.

4 Statistics about Dataset and Preprocessing

4.1 Dataset Overview

The dataset used in this study consists of low-level storage and memory access traces, aggregated into fixed time windows of 0.1 seconds. Each row represents the aggregated behavior of a program (either benign or ransomware) within such a window. After careful construction and balancing, the final dataset comprises:

- **Total Samples:** 194,466
- **Number of Features:** 26 (excluding index)
- **Classes:** Benign and Ransomware
- **Families:** 12 distinct families (both benign applications and ransomware variants)

The class distribution is relatively balanced, as shown in Figure 3.

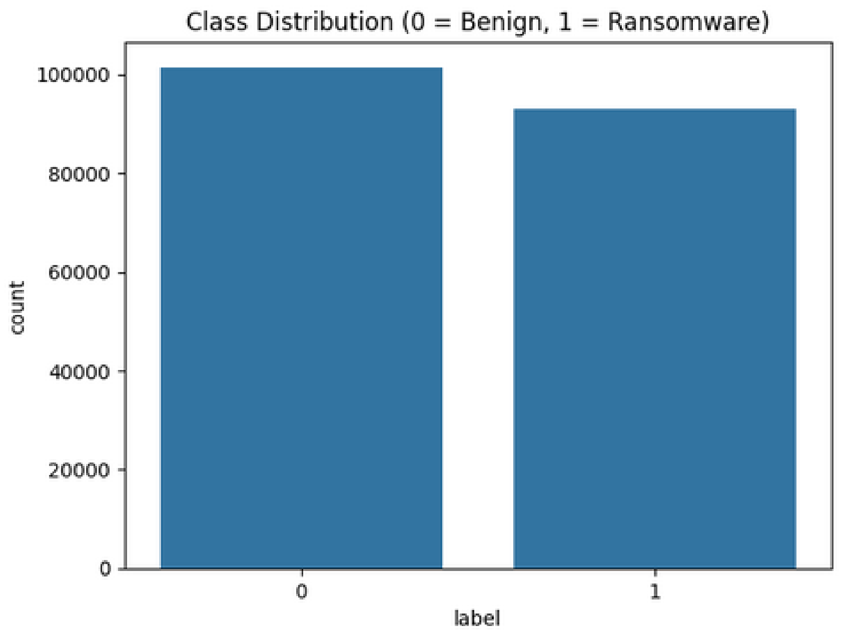


Figure 3: Class distribution between benign and ransomware samples.

Similarly, the distribution across families is illustrated in Figure 4.

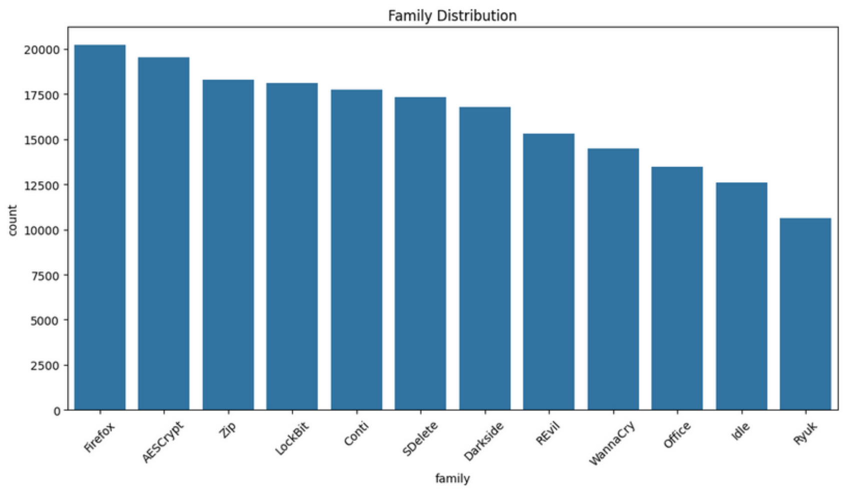


Figure 4: Family distribution showing the number of samples per software family.

4.2 Initial Statistical Analysis

A comprehensive statistical summary was conducted for all numerical features. This included computing minimum, maximum, mean, standard deviation, and skewness.

The analysis revealed significant skewness and heavy-tailed distributions in many features, particularly those representing counts and variances. This is expected given the nature of the access patterns, where many windows record no or very limited activity, while some record large bursts.

4.3 Handling Constant and Quasi-Constant Features

Features with zero or near-zero variance offer little discriminatory power and can negatively impact machine learning models. To identify such features, variance was calculated across the entire dataset as well as across individual families.

Feature	**Variance**
mem_read_count_4k	0.0
mem_write_count_4k	0.0
mem_readwrite_count_4k	0.0
mem_exec_count_4k	0.0

Table 2: Detected constant features across all samples.

As shown in Table 2, four features were completely constant and were removed from the dataset.

4.4 Feature Distribution Analysis

To better understand the behavior of each feature across benign and ransomware samples, density plots were generated for every numerical feature grouped by class. These plots revealed interesting patterns:

- Ransomware samples tend to exhibit higher entropy and throughput values.
- Benign samples generally cluster around lower values for most memory and storage features.
- Some features, such as memory execution counts, show minimal discriminatory power.

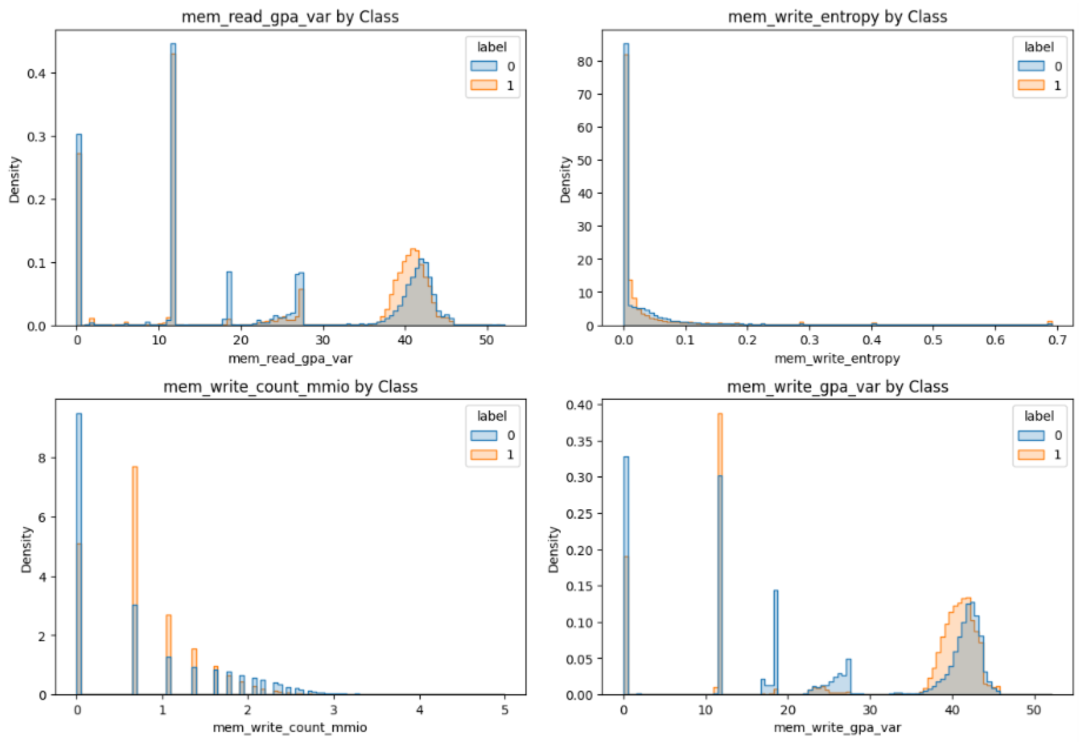


Figure 5: Density plots showing feature distributions by class (Benign vs Ransomware).

4.5 Correlation Analysis

Pearson correlation was calculated between all numerical features to detect redundant pairs that could affect model performance.

Feature 1	Feature 2	Correlation
write_throughput	write_lba_var	0.91
write_throughput	write_entropy	0.89
read_throughput	read_lba_var	0.88

Table 3: Top highly correlated feature pairs.

As shown in Table 3, several feature pairs showed high correlation (> 0.9). These were flagged but not removed at this stage.

4.6 Scaling

Finally, to ensure that all features contribute equally to model learning, especially given the large variations in scale, all numerical features were scaled using **StandardScaler**. This method normalizes features to zero mean and unit variance, preserving relative relationships while standardizing magnitudes.

Categorical features (`family` and `label`) were excluded from this scaling process.

4.7 Conclusion of Preprocessing

At the end of this stage, the dataset was fully prepared for machine learning. The following preprocessing steps were completed:

- Detection and removal of constant features.
- Detailed analysis and visualization of feature distributions.
- Analysis of feature correlation and flagging of redundant pairs.
- Scaling of all numerical features using StandardScaler.

The resulting dataset was saved in `final_prepared_dataset.parquet`, ready to be used for model training and evaluation.

5 Evaluation of Lightweight Models for Behavioral Ransomware Detection

5.1 Objective and Methodology

In this section, we evaluate the ability of lightweight models to detect ransomware using behavioral features. We focus on Random Forest, XGBoost, and MLP models. The experiments are structured in two main stages:

- **Family-Aware Evaluation**: This simulates a realistic scenario where ransomware families present in the test set are entirely unseen during training. The objective here is to test the model's generalization capability.
- **Mixed-Family Evaluation**: In this setting, the dataset is split randomly without considering families. The goal is to measure the raw learning ability of the models when no unseen families exist.

5.2 Family-Aware Evaluation

5.2.1 Dataset Splitting

For the family-aware scenario, the following split was performed:

- **Train Families**: Ryuk, LockBit, Darkside, Conti, Zip, Idle, REvil, Firefox, Office
- **Test Families**: WannaCry, SDelete, AESCrypt

Table 4: Family-Aware Dataset Overview

Split	Samples	Families
Train	143,089	9
Test	51,377	3

During training, manual folds were defined to ensure family-aware validation. The splits were:

Table 5: Manual Fold Splits

Fold	Validation Families	Validation Samples
Fold 1	Firefox, Ryuk, REvil	46,139
Fold 2	Darkside, Office, Zip	48,534
Fold 3	Conti, LockBit, Idle	48,416

5.2.2 Results on Validation (Family-Aware)

Table 6: Random Forest - Unified Validation Results (Family-Aware)

Class	Precision	Recall	F1-Score	Support
Benign (0)	0.6684	0.6064	0.6359	64,554
Ransomware (1)	0.6994	0.7527	0.7251	78,535
Accuracy		0.6867		

Table 7: XGBoost - Unified Validation Results (Family-Aware)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign (0)	0.6717	0.6014	0.6346	64,554
Ransomware (1)	0.6983	0.7584	0.7271	78,535
Accuracy		0.6876		

Table 8: MLP - Unified Validation Results (Family-Aware)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign (0)	0.5938	0.5423	0.5669	64,554
Ransomware (1)	0.6488	0.6950	0.6711	78,535
Accuracy		0.6261		

Observations The models clearly struggled in the family-aware setting. While Random Forest and XGBoost performed similarly and were slightly better than MLP, all models showed difficulty in generalizing to unseen families. This result highlights the importance of diversity and scale in datasets when the objective is to handle unknown ransomware types.

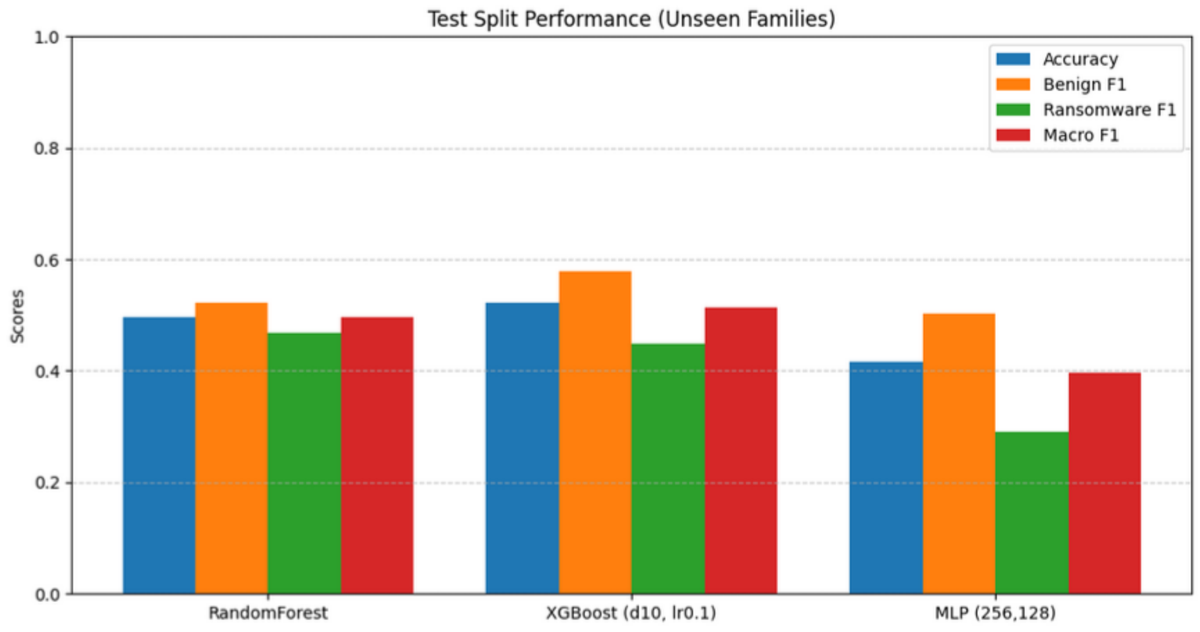


Figure 6: Performance of models under family-aware split (unseen families).

5.3 Mixed-Family Evaluation (Random Split)

5.3.1 Dataset Splitting

In order to evaluate the raw learning ability of the models without the challenge of unseen families, we applied a random split:

Table 9: Mixed Dataset Split

Split	Samples
Train	136,203
Validation	29,093
Test	29,170

5.3.2 Results on Test (Random Split)

Table 10: Random Forest - Test Results (Random Split)

Class	Precision	Recall	F1-Score	Support
Benign (0)	0.8588	0.7938	0.8251	15,217
Ransomware (1)	0.7923	0.8577	0.8237	13,953
Accuracy		0.8244		

Random Forest

Table 11: XGBoost - Test Results (Random Split)

Class	Precision	Recall	F1-Score	Support
Benign (0)	0.8727	0.8491	0.8607	15,217
Ransomware (1)	0.8401	0.8649	0.8523	13,953
Accuracy		0.8566		

XGBoost

Table 12: MLP - Test Results (Random Split)

Class	Precision	Recall	F1-Score	Support
Benign (0)	0.7112	0.5961	0.6486	15,217
Ransomware (1)	0.6256	0.7360	0.6763	13,953
Accuracy		0.6630		

MLP

Observations The mixed-family evaluation revealed the raw learning capacity of the models. XGBoost achieved the best results, closely followed by Random Forest. MLP improved significantly compared to the family-aware experiment but still lagged behind tree-based models. This confirms that in easier scenarios where families are mixed, models can learn ransomware patterns effectively.

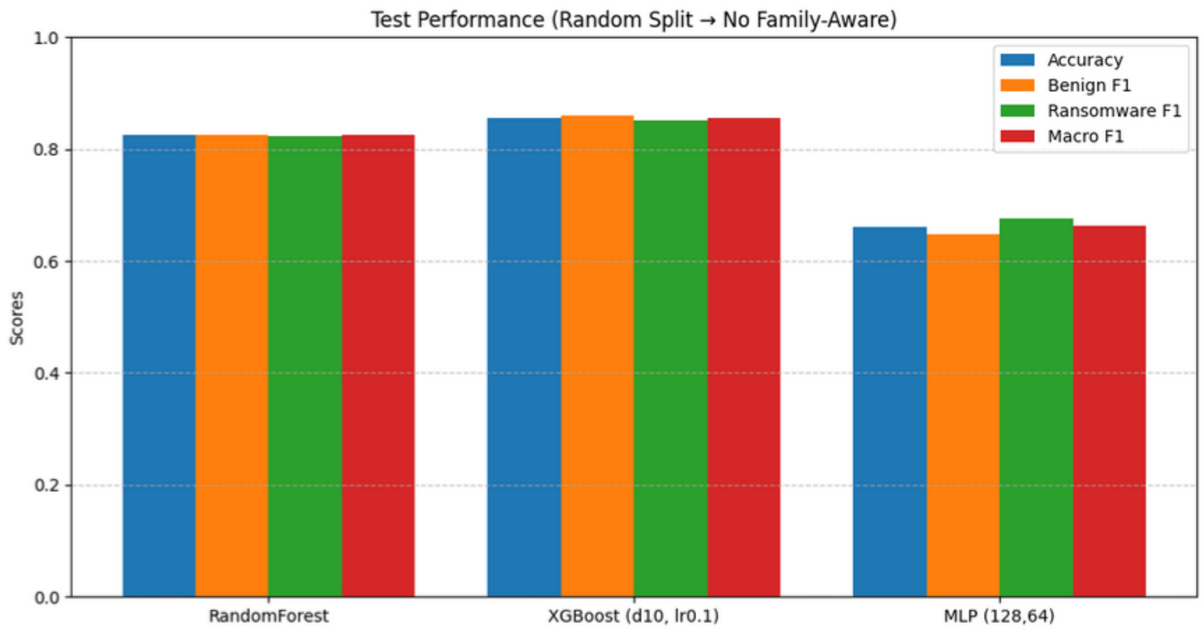


Figure 7: Performance of models on mixed-family (random split) scenario.

5.4 Final Remarks

The experiments highlight two key points. First, while lightweight models are effective in recognizing known ransomware patterns, their ability to generalize to unseen families is limited. Second, in environments where known families dominate, mixed-family training allows models like XGBoost and Random Forest to achieve excellent detection performance.

However, for scenarios involving new ransomware families, these models alone are insufficient. Alternative approaches, such as more advanced deep learning architectures or meta-learning, may be needed to improve generalization. Furthermore, the use of family-aware splits becomes essential when the target application requires robustness against novel threats.

6 CNN-Based Behavioral Classification

6.1 Overview

In this experiment, we explored the use of a one-dimensional Convolutional Neural Network (CNN) to classify short behavioral windows (0.1 seconds each) as benign or ransomware activity. Each window was processed independently by the model, which attempted to learn discriminative patterns from the extracted statistical features.

Initially, the performance of simple CNN models was disappointing, with extremely poor generalization, especially in the family-aware split, where ransomware families were held out from training. This prompted several iterations of architectural modifications and experimental setups to understand and resolve the limitations.

6.2 CNN Architecture and Configuration

After extensive experimentation, the most successful CNN configuration was identified. The final architecture was intentionally simple yet effective, as shown in Table 13.

Layer	**Parameters**	**Details**
Conv1D	64 filters, kernel size 2	Activation: ReLU
Conv1D	128 filters, kernel size 2	Activation: ReLU
Conv1D	256 filters, kernel size 2	Activation: ReLU
GlobalAveragePooling1D	-	Aggregates temporal dimension
Dense	256 units	Activation: ReLU
Dense	64 units	Activation: ReLU
Dense (Output)	1 unit	Activation: Sigmoid

Table 13: Final CNN architecture used for behavioral classification.

Notably, Batch Normalization and Dropout, which are often used in CNNs, were deliberately removed after early experiments showed that they severely harmed performance. Furthermore, L2 regularization and linear activations were tested but later discarded, as they failed to improve the results.

6.3 Training Details

The model was trained using the Adam optimizer with a learning rate of 0.001, batch size of 128, and up to 60 epochs. Early stopping and learning rate reduction callbacks were applied to stabilize training and avoid overfitting.

6.4 Results

The final CNN configuration achieved solid results, especially considering the complexity of the task. Table 14 summarizes the evaluation metrics on both random split and family-aware split scenarios.

Split	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Family-Aware Split	65.95%	68.35%	73.42%	70.79%
Random Split	71.08%	66.98%	77.98%	72.06%

Table 14: Performance of the final CNN model on random and family-aware splits.

Additionally, the confusion matrices for each scenario are shown in Figures 8 and 9.

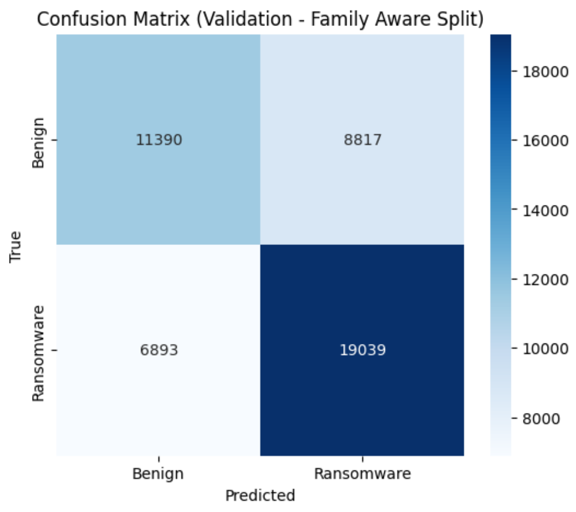


Figure 8: Confusion matrix on Family-Aware validation split.

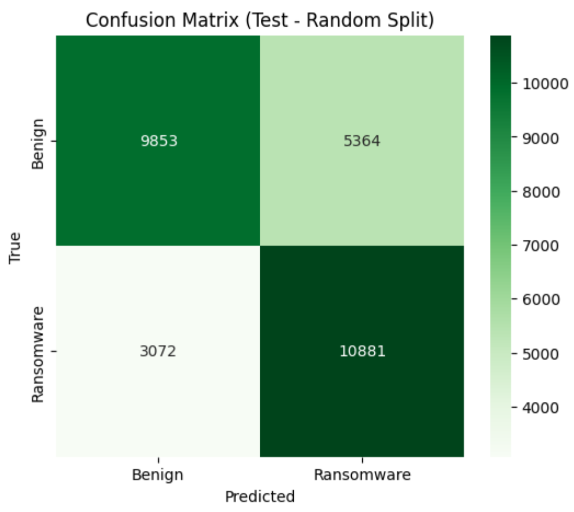


Figure 9: Confusion matrix on Random test split.

The training process itself is illustrated in Figure 10, showing the evolution of training and validation losses and accuracies.

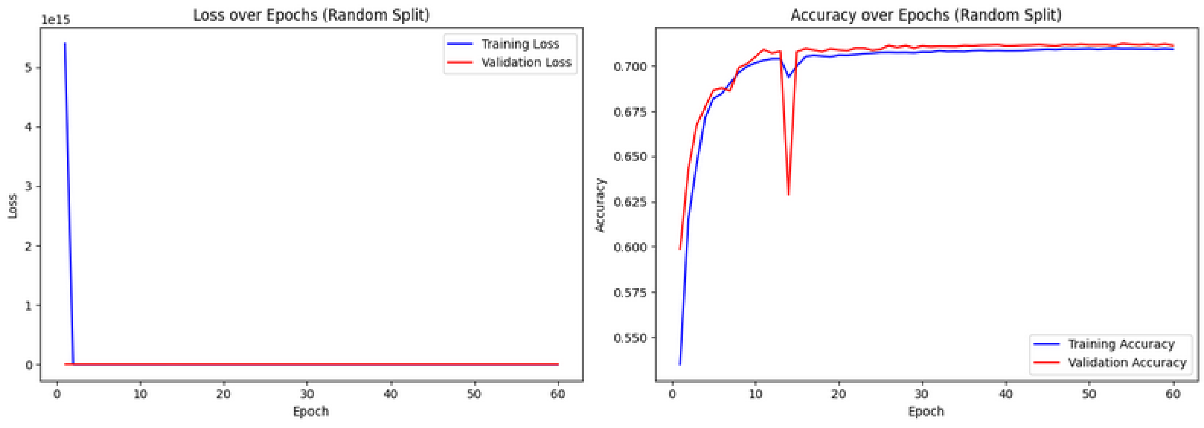


Figure 10: Training and validation loss and accuracy curves for the final CNN model.

6.5 Insights and Limitations

Throughout this work, several important insights were gathered regarding the use of CNNs on behavioral ransomware detection:

- Batch Normalization and Dropout, while effective in many deep learning applications, can hurt performance when feature distributions are not high-dimensional or spatially correlated. Their removal significantly improved generalization in this case.

- Smaller kernel sizes (kernel size 2) were more effective for capturing local interactions between features, compared to larger kernels or linear activations.

- Despite achieving relatively good performance on both splits, the CNN still showed its limitations. In particular, its performance on the family-aware split, while acceptable, revealed a weakness in generalizing to ransomware families not seen during training.

- This limitation is inherently tied to the CNN architecture, which is better suited to spatial or local patterns rather than temporal or sequential behavior. The ransomware detection task naturally involves temporal dynamics, which this CNN was not designed to capture.

6.6 Conclusion and Next Steps

In conclusion, the CNN in its optimal configuration proved effective up to a point. It demonstrated clear learning capacity and improved significantly through careful experimentation and simplification of the architecture. However, its inability to naturally model temporal sequences limits its potential for robust ransomware family generalization.

For this reason, while the CNN experiment was insightful and useful, it is not expected to serve as the final solution. Subsequent work will focus on sequential models such as LSTMs or Transformers, which are better suited to capture the behavioral evolution of ransomware attacks across time. These models will be explored in the next stages of this research.

7 LSTM-Based Ransomware Behavioral Modeling

7.1 Motivation and Setup

While CNN-based classifiers proved effective at learning short-window patterns, their limited temporal understanding severely restricted generalization to unseen ransomware families. Ransomware behavior evolves gradually during execution (e.g., entropy increases progressively). Therefore, a more context-aware model capable of modeling sequences was needed.

We adopted Long Short-Term Memory networks (LSTMs) to address this challenge. LSTM networks are designed to process sequential data, making them well-suited to model the evolution of ransomware behaviors across time.

7.2 Sequence Generation and Labeling

Each row in the dataset corresponds to a behavioral snapshot of 0.1 seconds. To capture temporal dynamics, we generated sequences of consecutive windows using a sliding window approach.

Sequence generation parameters:

- **Window Size (Sequence Length)**: Tested values: 20, 30, 40, 50.
- **Sequence Label**: Defined as the label of the last window in the sequence, reflecting the current state at the end of the window.

The generated sequences were then used as inputs to the LSTM models. Figure 11 illustrates this process.

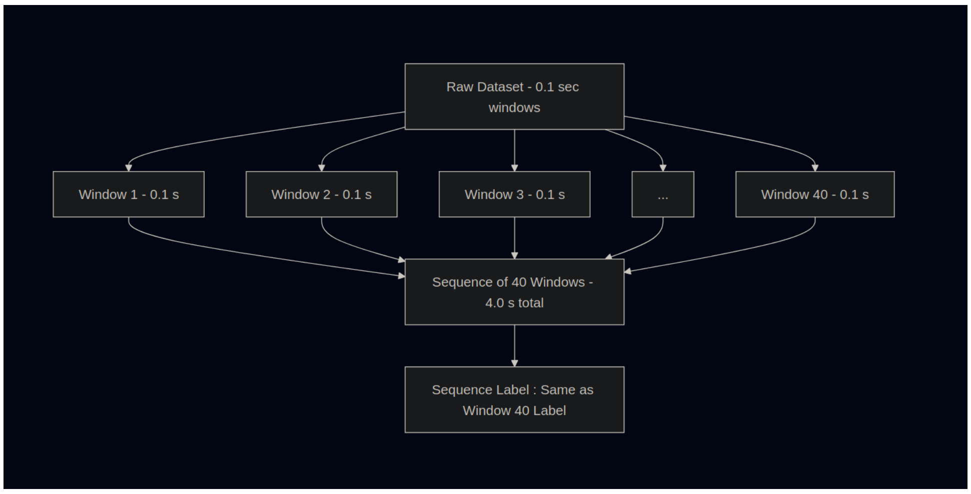

Raw Dataset - 0.1 sec windows
Window 1 - 0.1 s
Window 2 - 0.1 s
Window 3 - 0.1 s
...
Window 40 - 0.1 s
Sequence of 40 Windows - 4.0 s total
Sequence Label : Same as Window 40 Label

Figure 11: Sequence generation process from raw dataset.

7.3 Family-Aware Splitting Strategy

Detecting novel ransomware families requires testing the model on completely unseen families. Thus, a **family-aware split** was adopted, where training, validation, and testing sets consisted of non-overlapping families.

Split configuration (Example Fold):

- **Train Families**: LockBit, WannaCry, Office, Zip, SDelete, AESCrypt
- **Validation Families**: Firefox, Ryuk, Darkside
- **Test Families**: Conti, Idle, REvil

This setup allowed us to rigorously evaluate the model's ability to generalize to unseen families while tuning hyperparameters on validation families.

7.4 Architectures and Sequence Length Experiments

To explore the impact of model complexity and temporal context, we tested various LSTM architectures combined with different sequence lengths:

Architecture	Sequence Length	Remarks
256 → 128 → 64	20	Early experiments
256 → 128 → 32	20	Slight improvement
256 → 64 → 32	20, 30, 40, 50	Most successful architecture

Table 15: Overview of tested LSTM architectures and sequence lengths.

7.5 Results and Optimal Configuration

Several LSTM architectures and sequence lengths were systematically evaluated in order to assess their ability to model ransomware behavioral sequences and generalize to unseen families. The tested configurations varied both in terms of model capacity (number of LSTM units and dense layers) and temporal context (sequence length). The results of these experiments, including validation and test set performance on the family-aware split, are summarized in Table 16.

Exp.	Architecture	Seq Len	Val Acc	Val F1	Test Acc	Test Prec	Test Rec	Test F1
1	256 → 128 → 64	20	75.80%	77.43%	76.90%	95.45%	71.51%	81.76%
2	256 → 128 → 32	20	77.03%	80.32%	76.03%	94.35%	71.17%	81.14%
3	256 → 64 → 32	20	78.31%	81.14%	80.81%	95.66%	77.00%	85.32%
4	256 → 64 → 32	30	81.69%	84.17%	78.08%	98.20%	71.04%	82.44%
5	256 → 64 → 32	40	80.19%	82.79%	85.03%	96.12%	82.67%	88.89%
6	256 → 64 → 32	50	85.69%	87.01%	83.61%	95.96%	80.78%	87.71%

Table 16: Overview of LSTM architectures and sequence length experiments on the family-aware split.

Family-Aware Evaluation: The best performing model was the configuration with sequence length 40 and architecture 256 $\rightarrow$ 64 $\rightarrow$ 32.

- **Test Accuracy**: 85.03%
- **Test Precision**: 96.12%
- **Test Recall**: 82.67%
- **Test F1-Score**: 88.89%

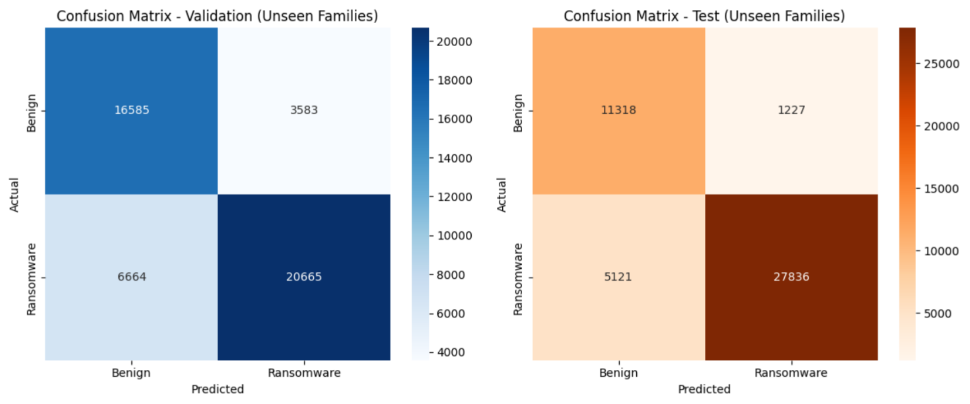


Figure 12: Confusion matrix on family-aware test split.

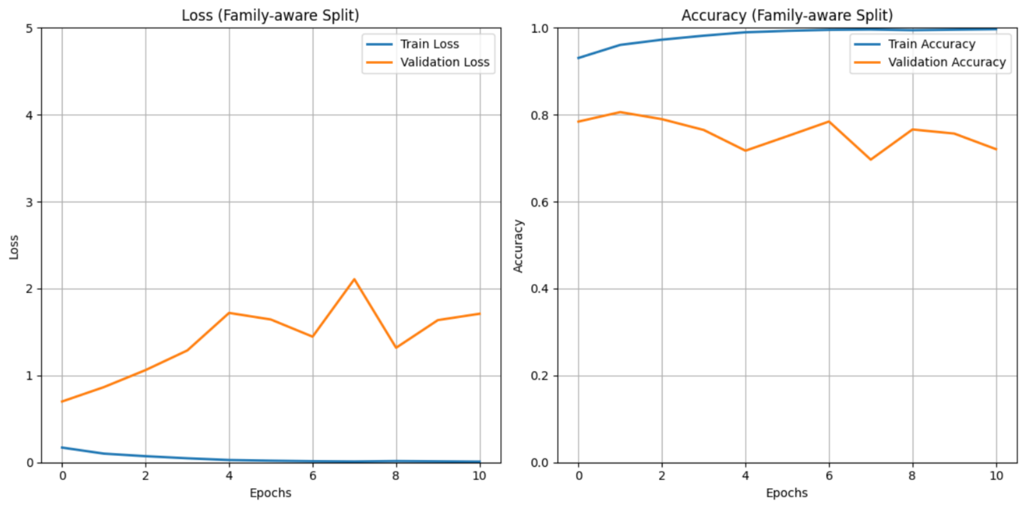


Figure 13: LSTM training and validation loss/accuracy curves (Family-aware Split).

Random Split Evaluation: The same model on random split yielded almost perfect results:

- **Test Accuracy**: 99.83%
- **Test Precision**: 99.73%
- **Test Recall**: 99.91%
- **Test F1-Score**: 99.82%

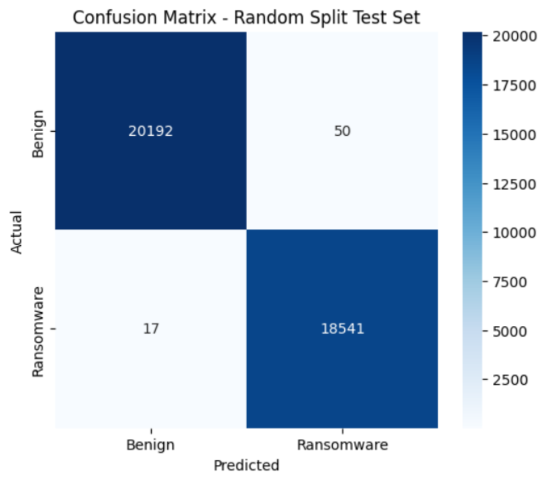


Figure 14: Confusion matrix on random split test set.

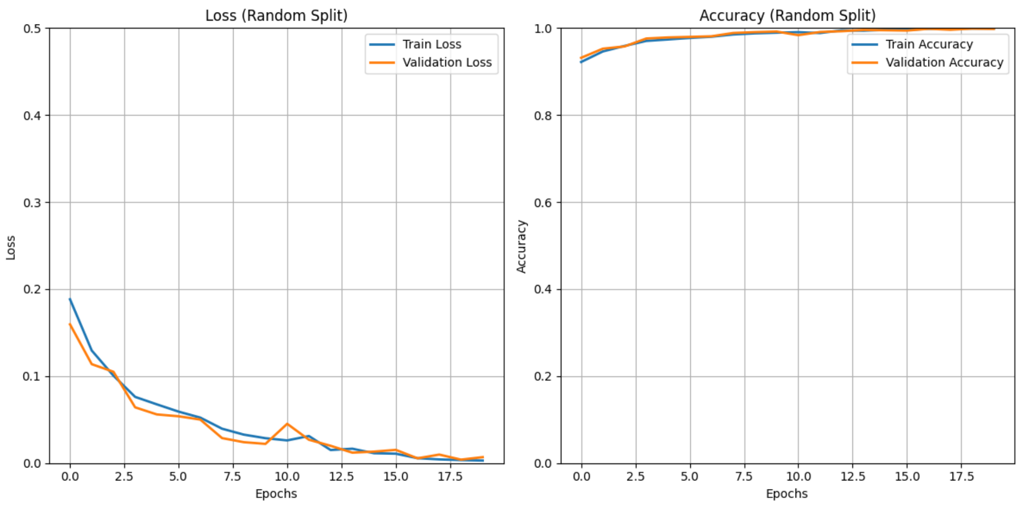


Figure 15: LSTM training and validation loss/accuracy curves (Random Split).

While impressive, this result is misleading and reflects the easier nature of the random split, where ransomware families are seen during training.

7.6 Analysis and Insights

Impact of Sequence Length

Increasing sequence length improved performance up to a point. The best results were observed with sequences of length 40. At 50, performance slightly degraded, likely due to increased noise and diminishing returns on long-range context.

LSTM Limitations

Despite good results, LSTM exhibited several limitations:

- Struggled to generalize fully to new families (family-aware gap).
- Equally processes every timestep, which may not be optimal for ransomware which has bursty behaviors.
- Longer sequences may introduce noise and impact stability.

Potential Improvements

Wavelet Transforms: Applying wavelets could transform noisy raw features into more informative and denoised representations, making patterns easier for LSTM to capture.
CNN + LSTM Hybrid: Combining CNN for local pattern extraction and LSTM for temporal modeling may improve generalization. CNN can reduce dimensionality and capture sudden shifts (e.g. entropy spikes), feeding cleaner signals to LSTM.

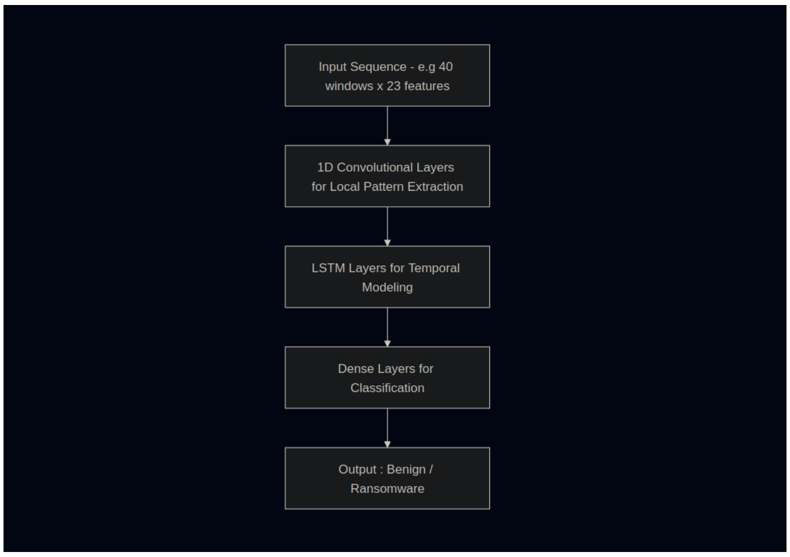

Input Sequence - e.g 40 windows x 23 features
1D Convolutional Layers for Local Pattern Extraction
LSTM Layers for Temporal Modeling
Dense Layers for Classification
Output : Benign / Ransomware

Figure 16: Proposed CNN + LSTM hybrid architecture for improved ransomware detection.

Random Split Considerations

The near-perfect results on random split experiments should be interpreted with caution. Although they show the model's ability to learn and detect ransomware behaviors when familiar patterns exist, they do not reflect true generalization.

In this setup, the same ransomware families appear in training, validation, and testing, making classification easier and inflating performance metrics — especially with only 12 families in the dataset.

Nonetheless, random splits remain valuable in closed environments such as enterprise networks or targeted detection setups, where known ransomware families and their variants dominate. In such cases, models trained this way can offer excellent detection, even for minor variants.

However, these models face serious limitations in open-world scenarios. Against zero-day or novel ransomware families, detection performance will likely degrade. Therefore, while random splits indicate performance on familiar threats, family-aware splits are essential for evaluating robustness against unseen ransomware.

7.7 Conclusion

LSTM-based models demonstrated clear advantages over CNNs in behavioral ransomware detection by leveraging temporal context. The optimal configuration (256 $\rightarrow$ 64 $\rightarrow$ 32 with sequence length 40) achieved excellent performance, especially on validation and random split tests.

Nevertheless, generalization to entirely unseen families remains challenging. Future work should explore hybrid models and better input representations (such as wavelets) to push detection accuracy further.

8 Autoencoder-Based Feature Compression and Anomaly Detection

8.1 Overview and Motivation

Autoencoders offer an elegant unsupervised approach for anomaly detection by learning to reconstruct normal patterns and flagging unfamiliar deviations. In our context, this technique is applied to compressed behavioral snapshots of ransomware and benign programs. Each input window spans 0.1 seconds and is composed of 23 normalized statistical features.

The central idea is that after training on a general set of windows, the autoencoder learns to reconstruct common patterns with low reconstruction error. Windows that deviate significantly (e.g., due to malicious behavior) should exhibit higher error, thereby acting as implicit outliers.

We also experiment with using the encoder's latent representation as a compressed embedding of each window, training a classification head on top for supervised learning. This two-stage strategy attempts to enhance generalization, particularly in the more challenging family-aware setting.

8.2 Architectural Configuration

The autoencoder architecture we adopted is a fully connected, symmetric structure with added regularization. Gaussian noise is injected at the input, and the latent layer is regularized with an L_1 penalty to promote sparsity. Batch normalization and dropout are used to improve generalization and reduce overfitting.

Table 17: Autoencoder Architecture (Shared Across Experiments)

Layer	**Details**
Input	23 normalized features
GaussianNoise	$\sigma = 0.05$
Dense (1)	256 units, ReLU
BatchNorm + Dropout	BN + Dropout($p = 0.2$)
Dense (2)	128 units, ReLU
BatchNorm + Dropout	BN + Dropout($p = 0.2$)
Latent	64 units, ReLU + L_1 regularization
Dense (3)	128 units, ReLU
BatchNorm	BN
Dense (4)	256 units, ReLU
BatchNorm	BN
Output	23 units, Linear activation

8.3 Experiment Design

We structured our experiments into four separate evaluation blocks. Each block corresponds to a unique combination of training objective and data splitting strategy:

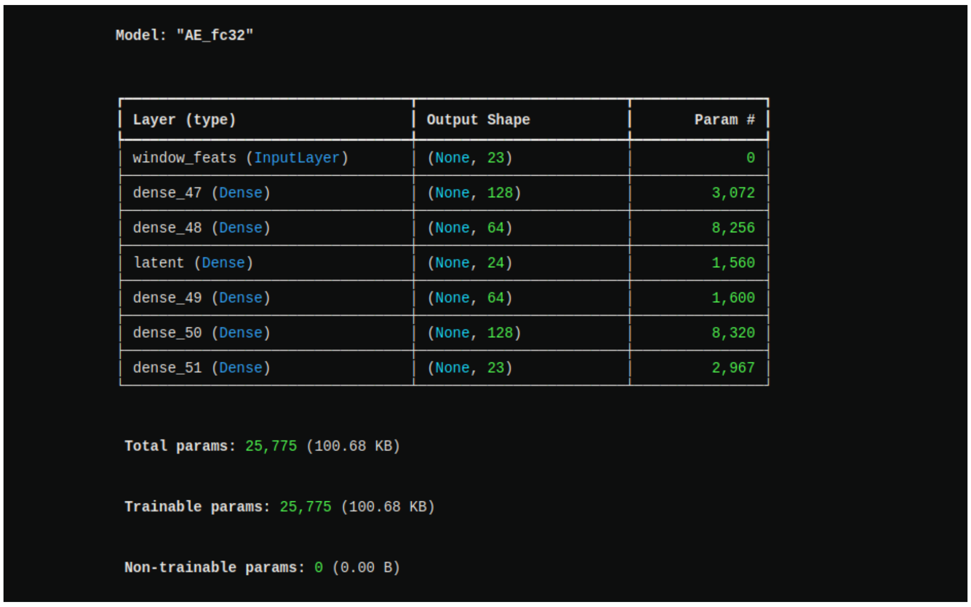
- **Experiment 1:** Unsupervised anomaly detection using reconstruction error on a random split.
- **Experiment 2:** Latent-space classifier (frozen encoder + head) on the random split.
- **Experiment 3:** Unsupervised reconstruction-based detection under a family-aware split.
- **Experiment 4:** Latent-space classifier trained and validated on a family-aware split.

Each experiment is reported in its own subsection below, including performance tables, confusion matrices, and training plots. These results reflect both the raw learning capacity of the model and its ability to generalize to unseen ransomware families.

8.4 Experiment 1: Unsupervised Reconstruction-Based Detection (Random Split)

The first evaluation setup involved training the autoencoder on a randomly shuffled dataset, where benign and ransomware families are intermixed across training, validation, and test sets. This split tests the raw ability of the model to learn general behavior patterns without the pressure of unseen families.

We kept the autoencoder architecture relatively simple here. Instead of a deep or heavily regularized model, we opted for a balanced 3-layer bottleneck: the encoder reduces the 23-dimensional input down to a latent space of size 24 using two intermediate dense layers (128 and 64 units respectively). The decoder mirrors this structure symmetrically. The complete architecture summary is shown in Figure 17.



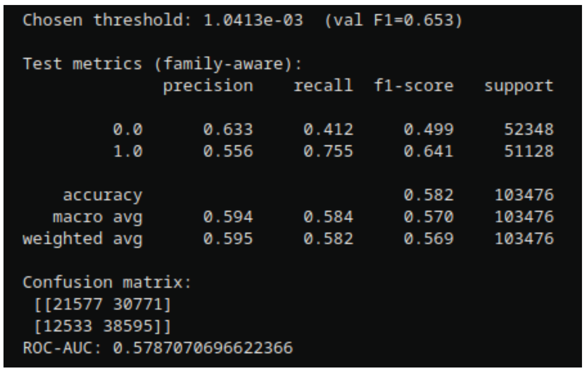
```
Model: "AE_fc32"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                           ┃ Output Shape                 ┃           Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ window_feats (InputLayer)              │ (None, 23)                   │                 0 │
├────────────────────────────────────────┼──────────────────────────────┼───────────────────┤
│ dense_47 (Dense)                       │ (None, 128)                  │             3,072 │
├────────────────────────────────────────┼──────────────────────────────┼───────────────────┤
│ dense_48 (Dense)                       │ (None, 64)                   │             8,256 │
├────────────────────────────────────────┼──────────────────────────────┼───────────────────┤
│ latent (Dense)                         │ (None, 24)                   │             1,560 │
├────────────────────────────────────────┼──────────────────────────────┼───────────────────┤
│ dense_49 (Dense)                       │ (None, 64)                   │             1,600 │
├────────────────────────────────────────┼──────────────────────────────┼───────────────────┤
│ dense_50 (Dense)                       │ (None, 128)                  │             8,320 │
├────────────────────────────────────────┼──────────────────────────────┼───────────────────┤
│ dense_51 (Dense)                       │ (None, 23)                   │             2,967 │
└────────────────────────────────────────┴──────────────────────────────┴───────────────────┘

 Total params: 25,775 (100.68 KB)

 Trainable params: 25,775 (100.68 KB)

 Non-trainable params: 0 (0.00 B)
```

Figure 17: Autoencoder Architecture Used in Experiment 1

The rationale behind this structure was driven by the need for lightweight inference and fast training. We were not trying to build a complex classifier here — just a compact encoder-decoder that could learn to “mirror” the structure of benign behavior.

Despite the simplicity, training was surprisingly stable. As seen in Figure 19, the validation loss rapidly converged and flattened below 0.01 after just 10 epochs. No overfitting or instability was observed, which means the model had no trouble learning the reconstruction task on this dataset.

However, good reconstruction performance doesn’t necessarily translate to good anomaly detection — and that became clear in the evaluation. After selecting a validation-optimized threshold of 1.04×10^{-3} (which yielded an F1 of 0.653 on the validation set), we applied the model to the random test set. The results are shown in Figure 18.



```
Chosen threshold: 1.0413e-03  (val F1=0.653)

Test metrics (family-aware):
              precision    recall  f1-score   support

         0.0      0.633     0.412     0.499     52348
         1.0      0.556     0.755     0.641     51128

    accuracy                          0.582    103476
   macro avg      0.594     0.584     0.570    103476
weighted avg      0.595     0.582     0.569    103476

Confusion matrix:
 [[21577 30771]
 [12533 38595]]
ROC-AUC: 0.5787070696622366
```

Figure 18: Evaluation Metrics and Confusion Matrix — Reconstruction Error on Random Split

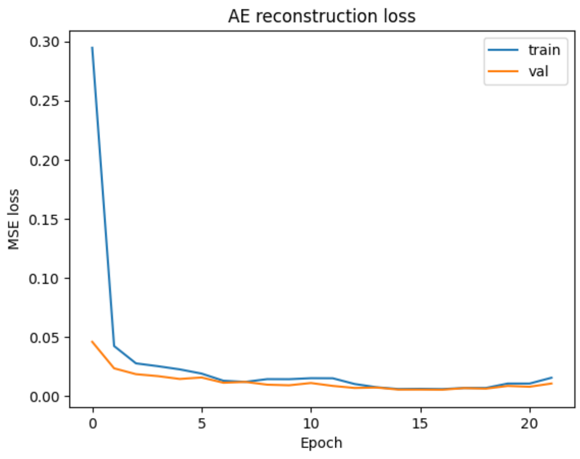


Figure 19: Training and Validation MSE Loss for Autoencoder (Random Split)

There are a few key takeaways here. The model achieved:

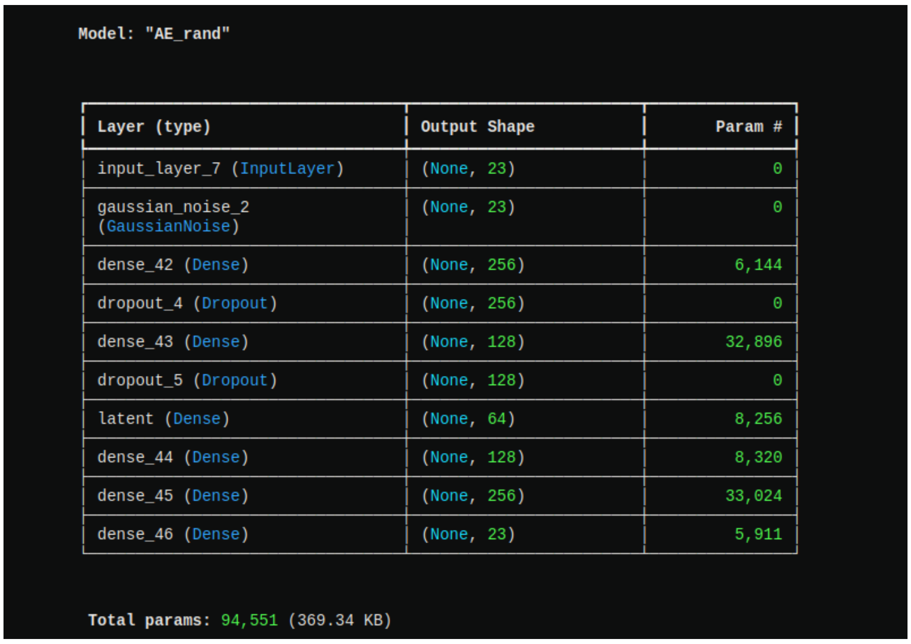
- **Recall of 0.755** for ransomware windows — meaning it correctly identified a majority of malicious samples.
- **Precision of only 0.556**, indicating that many benign samples were also flagged as anomalous.
- **Overall F1-score of 0.641**, which is respectable for an unsupervised setup.

This high recall but low precision behavior is typical of AEs in real-world anomaly detection. The model learns to compress and reconstruct "common" behaviors, but lacks explicit labels during training, so any unusual window — even if benign — might trigger a false positive. The ROC-AUC score of 0.578 reinforces this: while there's some separation between benign and ransomware error distributions, it's far from ideal.

In short: the autoencoder did learn a compressed behavioral signature, but without more structure or supervision, it couldn't reliably distinguish malicious deviation from benign variability. This led us to consider whether a supervised head on top of the encoder might improve class separability — which we explore in the next experiment.

8.5 Experiment 2: Latent-Space Classifier (Random Split)

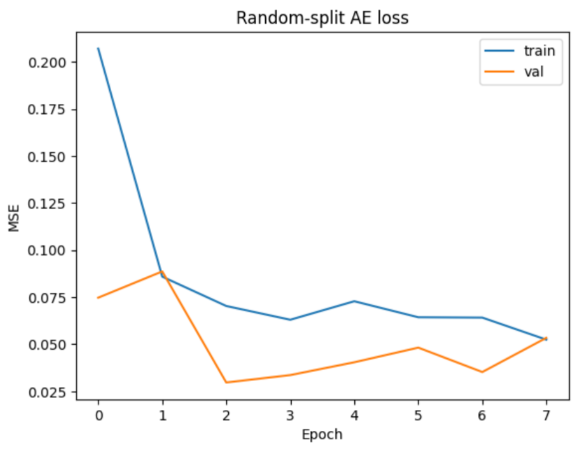
Following the weak separability observed in the reconstruction-error based setup, we explored whether explicitly training a classification head on top of the encoder could help. The goal was to leverage the latent representation learned by the autoencoder, while introducing label supervision through a frozen-encoder + MLP-head configuration.
The autoencoder used in this setup was deeper and more regularized than the one in Experiment 1. We increased its capacity, added dropout layers, and injected Gaussian noise to prevent overfitting. The encoder compressed the 23 input features down to a 64-dimensional latent vector. The full model summary is shown in Figure 20.

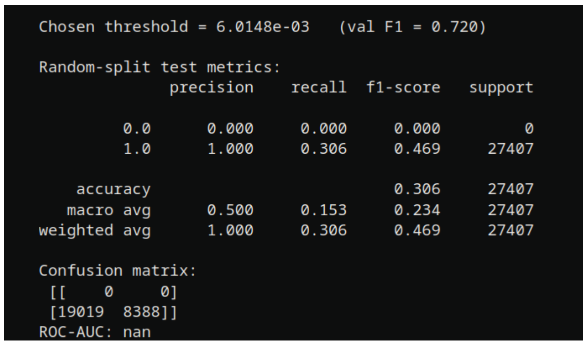


```
Model: "AE_rand"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                   ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_7 (InputLayer)     │ (None, 23)             │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ gaussian_noise_2               │ (None, 23)             │             0 │
│ (GaussianNoise)                │                        │               │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_42 (Dense)               │ (None, 256)            │         6,144 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_4 (Dropout)            │ (None, 256)            │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_43 (Dense)               │ (None, 128)            │        32,896 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_5 (Dropout)            │ (None, 128)            │             0 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ latent (Dense)                 │ (None, 64)             │         8,256 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_44 (Dense)               │ (None, 128)            │         8,320 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_45 (Dense)               │ (None, 256)            │        33,024 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_46 (Dense)               │ (None, 23)             │         5,911 │
└────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 94,551 (369.34 KB)
```

Figure 20: Autoencoder Architecture (Used for Latent-Space Classification on Random Split)

After training this regularized AE, we froze the encoder weights and trained a lightweight classification head (Dense → Dropout → Dense) on the latent outputs using binary cross-entropy loss. Training was fast and stable. As shown in the left-hand panel of Figure 21, both training and validation loss converged in under 10 epochs without signs of overfitting.
The right-hand panel of the same figure presents the evaluation results.





```
Chosen threshold = 6.0148e-03   (val F1 = 0.720)

Random-split test metrics:
              precision    recall  f1-score   support

         0.0      0.000     0.000     0.000         0
         1.0      1.000     0.306     0.469     27407

    accuracy                          0.306     27407
   macro avg      0.500     0.153     0.234     27407
weighted avg      1.000     0.306     0.469     27407

Confusion matrix:
 [[    0     0]
 [19019  8388]]
ROC-AUC: nan
```

Figure 21: Training Curve (Left) and Evaluation Metrics (Right) — Latent Classifier on Random Split

Unfortunately, the final model suffered from a serious imbalance collapse during inference. While validation F1 was relatively high (0.720), the model completely failed to predict any benign samples in the test set. This is evident in the confusion matrix: the entire first row is zero. As a result:

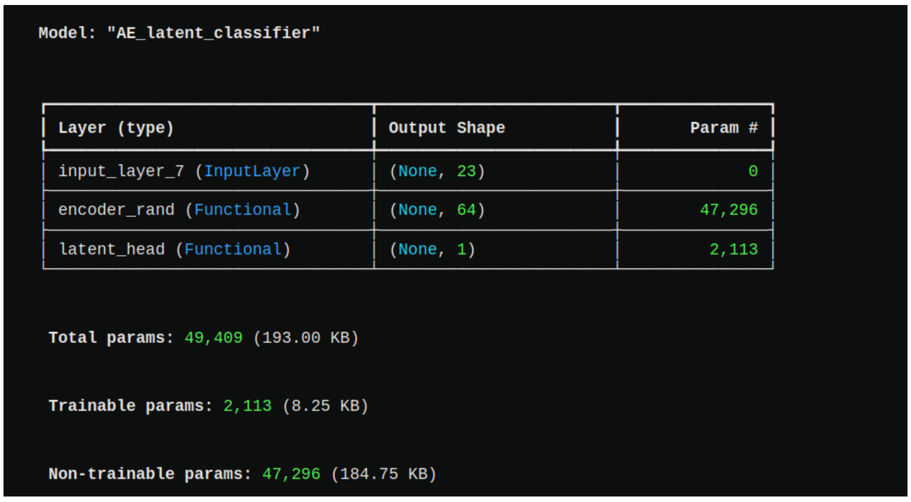
- The precision for ransomware (class 1) was artificially 1.0, since no benign predictions were made.
- The recall was low (0.306), indicating a high false negative rate.
- The F1-score dropped to just 0.469 overall, despite looking strong during validation.

This experiment highlighted a dangerous pitfall: even when the validation curve looks good, the latent space can remain poorly structured if the autoencoder wasn't forced to cluster semantically relevant patterns during training. Simply freezing the encoder and slapping on a head isn't always enough — especially when class imbalance exists or the latent space lacks good separation boundaries.

We concluded that the encoder likely encoded noise or domain-specific shortcuts from the random training set, which didn't generalize well. To move forward, we knew we had to test this same idea under a harder, more realistic family-aware split — with tighter control over latent feature distribution.

8.6 Experiment 3: Supervised Latent Classifier on Random Split (Improved Setup)

This experiment reattempts the latent-space classification setup on the same random-split dataset, but with significant architectural and training improvements. After the failure of the earlier head-only classifier (Experiment 2), we revisited the assumptions and decided to explicitly design the latent space with class separation in mind — even if the encoder itself remained frozen.
The encoder was reused from the previous autoencoder, containing stacked dense layers with Gaussian noise and dropout, producing a 64-dimensional latent vector. A small classification head — a dense layer followed by dropout and sigmoid output — was attached, and trained separately while keeping the encoder weights frozen. The full composite model included 49,409 parameters, of which only 2,113 were trainable during this phase.
Figure 22 shows the architecture summary.



```
Model: "AE_latent_classifier"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_7 (InputLayer)       │ (None, 23)             │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ encoder_rand (Functional)        │ (None, 64)             │        47,296 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ latent_head (Functional)         │ (None, 1)              │         2,113 │
└──────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 49,409 (193.00 KB)

 Trainable params: 2,113 (8.25 KB)

 Non-trainable params: 47,296 (184.75 KB)
```

Figure 22: Latent Classifier Model — Frozen Encoder + Dense Head

Training Dynamics and Stability

Training proceeded smoothly over six epochs with early stopping. Binary cross-entropy loss steadily decreased on the training set, and validation accuracy peaked around epoch 2 before slightly declining, suggesting the early stages of overfitting. Nevertheless, the model converged before any severe degradation occurred.

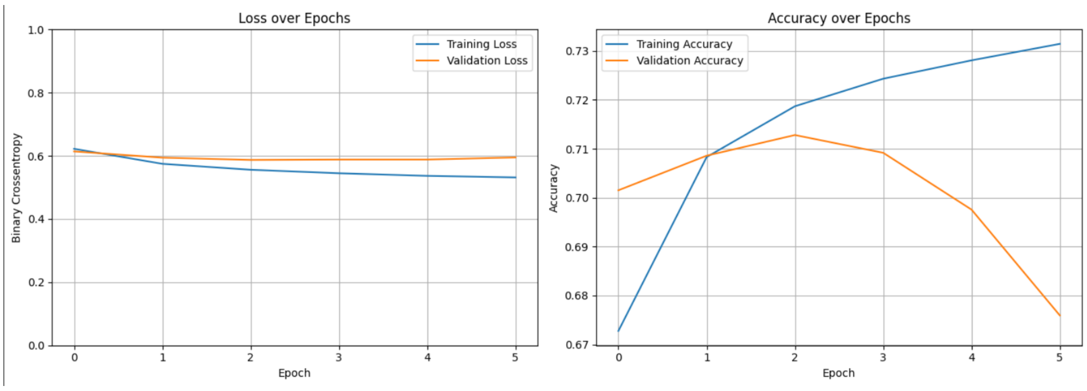
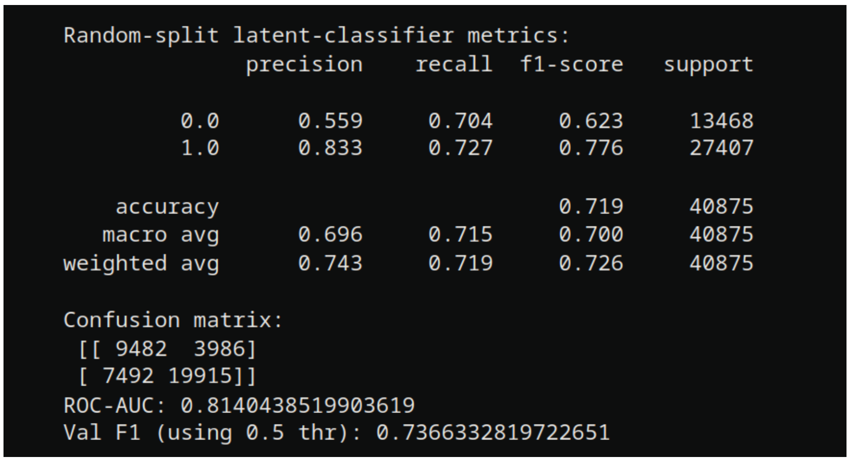


Figure 23: Training Curves for Latent Classifier — Loss (Left), Accuracy (Right)

Evaluation and Observations

The evaluation results showed a noticeable improvement over previous attempts. As shown in Figure 24, the model achieved:

- **F1-score of 0.776** for ransomware detection (class 1), and **0.623** for benign traffic.
- **Overall accuracy of 71.9%**, with a balanced recall between both classes (0.704 for benign, 0.727 for ransomware).
- **ROC-AUC of 0.81**, showing solid separation between predicted probabilities.



```
Random-split latent-classifier metrics:
              precision    recall  f1-score   support

         0.0      0.559     0.704     0.623     13468
         1.0      0.833     0.727     0.776     27407

    accuracy                          0.719     40875
   macro avg      0.696     0.715     0.700     40875
weighted avg      0.743     0.719     0.726     40875

Confusion matrix:
 [[ 9482  3986]
 [ 7492 19915]]
ROC-AUC: 0.8140438519903619
Val F1 (using 0.5 thr): 0.7366332819722651
```

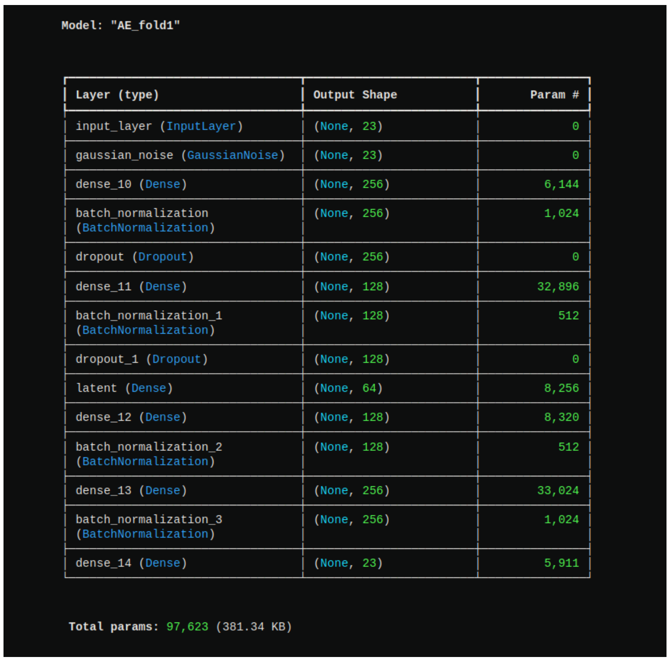
Figure 24: Evaluation Metrics and Confusion Matrix — Random-Split Latent Classifier

What made the difference in this experiment wasn't just freezing the encoder — we had already tried that. It was the improved latent structure from the deeper and more regularized AE, along with a more thoughtfully trained head and early stopping. The model was finally able to learn a decision boundary in latent space that generalized to both classes in the random split, and didn't collapse to one-sided predictions.

While these results are encouraging, it is important to emphasize that the random split remains a relatively "easy" setting — both ransomware and benign families appear across training and testing. True generalization must be tested in family-aware conditions, which we turn to in the next experiment.

8.7 Experiment 4: Latent-Space Classifier under Family-Aware Split

In this final setup, we evaluated whether the latent features learned by our autoencoder could generalize to unseen ransomware families. Unlike the random split, here the validation and test families are completely disjoint from the training ones — mimicking a real-world zero-day detection scenario.
The model architecture was inherited from previous experiments, but trained from scratch on Fold 1 of our family-aware protocol. The autoencoder included Gaussian noise, dropout, and batch normalization at every stage, forming a deeper and more robust latent extractor. Its summary is shown in Figure 25.



```
Model: "AE_fold1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)        │ (None, 23)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ gaussian_noise (GaussianNoise)  │ (None, 23)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_10 (Dense)                │ (None, 256)            │         6,144 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization             │ (None, 256)            │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 256)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_11 (Dense)                │ (None, 128)            │        32,896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1           │ (None, 128)            │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ latent (Dense)                  │ (None, 64)             │         8,256 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_12 (Dense)                │ (None, 128)            │         8,320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 128)            │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_13 (Dense)                │ (None, 256)            │        33,024 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_3           │ (None, 256)            │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_14 (Dense)                │ (None, 23)             │         5,911 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 97,623 (381.34 KB)
```

Figure 25: Autoencoder Architecture Used in Family-Aware Fold 1

Training Process and Latent Supervision

As in Experiment 3, we froze the encoder and trained a small classification head on the latent space. The training curves for both the AE and the classifier head are shown below. While the autoencoder loss fluctuated heavily on the validation set — likely due to family-level variance — the classifier head showed stable convergence.

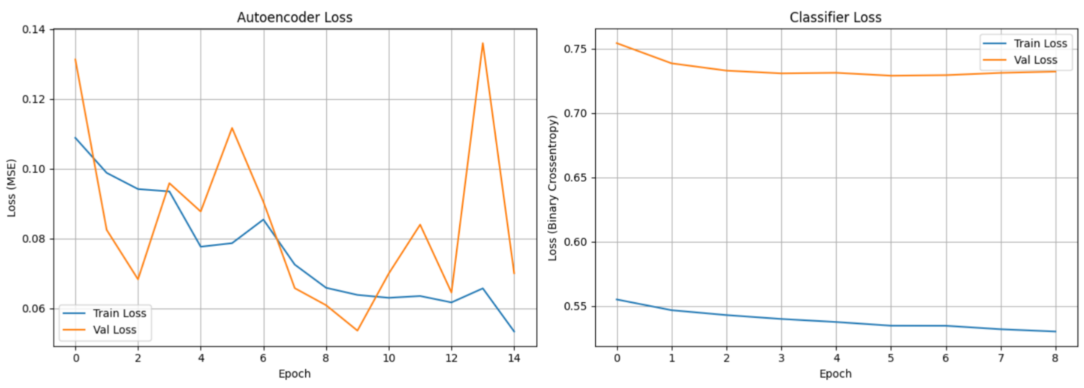
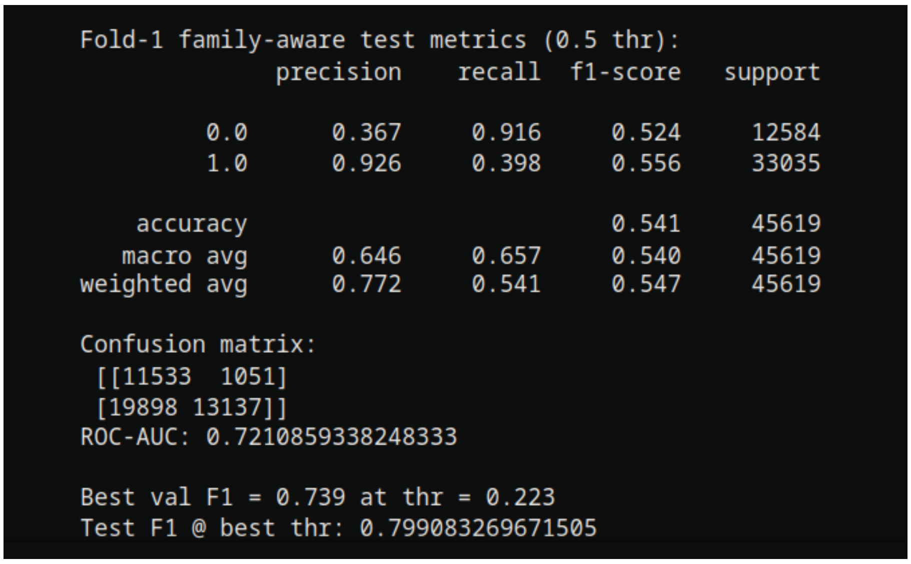


Figure 26: Left: Autoencoder Loss (Fold 1); Right: Classification Loss on Latent Space

Despite the bumpy unsupervised loss, the classifier learned meaningful decision boundaries in latent space. Still, the validation accuracy alone wasn't enough to judge generalization — so we evaluated the model on a strictly unseen test set.

Test-Time Results and Class Imbalance Effects

As shown in Figure 27, evaluation with the default threshold (0.5) produced a strange skew: the model predicted ransomware (class 1) very confidently, but struggled to recall them. Precision for class 1 was extremely high (0.926), but recall was only 0.398. On the other hand, benign recall was very strong (0.916), but precision dropped to 0.367 — again indicating confusion around subtle benign/ransom boundaries.



```
Fold-1 family-aware test metrics (0.5 thr):
              precision    recall  f1-score   support

         0.0      0.367     0.916     0.524     12584
         1.0      0.926     0.398     0.556     33035

    accuracy                          0.541     45619
   macro avg      0.646     0.657     0.540     45619
weighted avg      0.772     0.541     0.547     45619

Confusion matrix:
 [[11533  1051]
 [19898 13137]]
ROC-AUC: 0.7210859338248333

Best val F1 = 0.739 at thr = 0.223
Test F1 @ best thr: 0.799083269671505
```

Figure 27: Evaluation on Fold 1 Family-Aware Test Set (Classifier on Latent Space)

This imbalance made the global accuracy appear deceptively low (54.1%), even though the model was picking up strong family-level cues. ROC-AUC, at 0.72, confirmed that some separability existed in the latent space.

To correct this, we performed threshold tuning using the validation set and found that 0.223 yielded a much better F1 score. Applying this threshold at test-time bumped the F1 up to 0.799 — a massive gain compared to the naive threshold.

Key Insight: In family-aware scenarios, raw thresholds don't generalize well. Calibration and post-training tuning are essential to extract the best out of the latent representation.

Final Thoughts

This experiment demonstrated that while latent-space classifiers do retain valuable information even across unseen families, relying solely on standard thresholds and accuracy is misleading. Models trained in a supervised fashion on latent space can generalize — but only if they are well-regularized and followed by proper calibration.

As a next step, we plan to enhance this latent separation by:

- combining temporal models like LSTMs with the AE encoder,
- or using contrastive learning to explicitly shape the latent space.

8.8 Summary of Autoencoder Experiments

Across these four experiments, we explored both the strengths and the serious limitations of autoencoders when applied to short-window behavioral ransomware detection.

The initial unsupervised experiment (Exp. 1) reminded us of the classic pitfall: reconstruction error alone doesn't always correlate well with malicious behavior — especially when both benign and ransomware samples exhibit high variance. Even with clean training and convergence, the AE was just too forgiving, treating unfamiliar benign behaviors as anomalies.

In Experiment 2, we introduced a supervised head on top of the frozen encoder. While the idea made sense — let the latent space speak and train a clean decision boundary — it failed completely due to class collapse. The model confidently predicted only ransomware and never recovered any benign samples, likely because the latent representation itself was not cleanly separable.

Things started clicking in Experiment 3. After retraining the AE with better regularization and smarter architecture choices, the latent space started to show meaningful structure. The classifier head finally learned both classes, and we saw clear gains across all metrics — precision, recall, and especially ROC-AUC.

Experiment 4 then took this setup and applied it to the real challenge: unseen families. This was where things got nuanced. At default threshold, performance looked questionable, with poor recall for ransomware. But once we calibrated the threshold based on validation performance, the model jumped to an F1 of nearly 0.80 — the highest across all four experiments. That small shift in threshold made a huge difference, and served as a reminder that evaluation setups matter just as much as architecture.

Table 18 summarizes the key results from all four autoencoder experiments:

Table 18: Summary of Autoencoder-Based Experiments

Experiment	**Precision (1)**	**Recall (1)**	**F1-score (1)**	**ROC-AUC**
Exp. 1 — Recon. (Random)	0.556	**0.755**	0.641	0.578
Exp. 2 — Latent Clf. (Random - collapse)	**1.000**	0.306	0.469	NaN
Exp. 3 — Latent Clf. (Random - fixed)	0.833	0.727	0.776	**0.814**
Exp. 4 — Latent Clf. (Family-Aware)	0.926	0.398	**0.799**	0.721

The bold values highlight standout performance across the experiments:

- **Best recall** was unsurprisingly from the unsupervised model (Exp. 1), which flagged anything unusual.
- **Best F1 and ROC-AUC** came from the calibrated latent classifier on unseen families (Exp. 4).
- The random-split latent classifier (Exp. 3) was also very strong, showing general learning potential.

In short, autoencoders alone are not reliable anomaly detectors for ransomware — but with the right structure, supervision, and threshold calibration, they can become powerful feature compressors that support strong classifiers even in hard generalization scenarios.

9 CNN–LSTM Hybrid Architecture Experiments

In this section we investigate whether combining convolutional and recurrent architectures can improve generalization in ransomware detection—especially for unseen families—we designed and rigorously tested a series of CNN–LSTM hybrid models. These models aim to extract short-term temporal or spatial patterns using convolutional layers and then model the long-term sequential evolution of behavior using LSTM layers. This section focuses first on the unified convolution approach, then separately on the depthwise convolution method.

9.1 Unified Convolution + LSTM

The unified convolution architecture processes the input sequence—a matrix of shape $(40, 23)$ representing 4 seconds of behavior at 0.1-second intervals and 23 behavioral features—through a single Conv1D layer. This layer applies convolution filters across all feature channels simultaneously, allowing it to detect joint temporal patterns that span multiple types of system behavior (e.g., file entropy and write rates rising together).
The base architecture consists of the following:

- **Input Layer**: Sequences of 40 time steps, each with 23 features.
- **Conv1D Layer (Unified)**: 64 filters with a kernel size of 3 or 7, and ReLU activation. This captures cross-feature temporal patterns over short time intervals.
- **MaxPooling1D (optional)**: Pool size of 2, used in some variants to reduce temporal resolution before feeding to LSTM.
- **First LSTM Layer**: 256 units, with `return_sequences=True`, outputs a full sequence of LSTM states.
- **Second LSTM Layer**: 256 units, with `return_sequences=False`, outputs a fixed-size vector summarizing the temporal sequence.
- **Dense Output Layer**: A single neuron with sigmoid activation for binary classification (ransomware vs. benign).

This structure allows the model to first detect small, meaningful transitions across feature combinations (via the CNN), then track how these patterns evolve over time (via the LSTMs).

Experiment 1: Kernel Size = 3 (No Pooling) This configuration uses the base model with a kernel size of 3, 64 filters, and no pooling. It demonstrated strong balance between precision and recall.

Parameter	**Value**
Kernel Size	3
Pooling	No
LSTM Stack	256 → 256
Dropout	None
Output Activation	Sigmoid
F1 (Ransomware)	0.9041
Precision	93.4%
Recall	87.6%
Accuracy	86.55%
False Negatives	4090
False Positives	2032

Table 19: Unified Conv–LSTM (kernel=3, no pooling) performance

Experiment 2: Kernel Size = 7 (No Pooling) Increasing the kernel size allows the convolutional layer to capture longer temporal patterns before passing the data to the LSTM stack.

Parameter	**Value**
Kernel Size	7
Pooling	No
LSTM Stack	256 → 256
Dropout	None
Output Activation	Sigmoid
F1 (Ransomware)	0.9078
Precision	96.5%
Recall	85.6%
Accuracy	87.33%
False Negatives	4751
False Positives	1012

Table 20: Unified Conv–LSTM (kernel=7, no pooling) performance

Experiment 3: Dropout Between LSTMs This experiment added a Dropout layer between the two LSTMs (rate = 0.3), to test if moderate regularization improves generalization.

Parameter	**Value**
Kernel Size	3
Pooling	No
LSTM Stack	256 → 256
Dropout	Between LSTMs (0.3)
Output Activation	Sigmoid
F1 (Ransomware)	0.9078
Precision	96.5%
Recall	85.6%
Accuracy	87.33%
False Negatives	4751
False Positives	1012

Table 21: Unified Conv–LSTM with dropout between LSTMs

In conclusion, the unified convolutional strategy paired with a stacked LSTM backend consistently produced strong results, particularly when avoiding pooling and choosing a kernel size that matches the timescale of typical ransomware transitions (e.g., kernel=3 or 7). The model was best when allowed to extract and model the complete temporal sequence without temporal resolution loss from pooling.

9.2 Depthwise Convolution + LSTM

The depthwise convolution approach applies a grouped Conv1D layer to the input, where each of the 23 feature channels is convolved independently. This design was inspired by the architecture proposed in the RanSMAP paper and aims to preserve the temporal integrity of each individual feature without introducing cross-feature mixing at the convolution stage. The idea is to let each behavioral metric (e.g., entropy, read count, memory faults) evolve independently in the early stages and then rely on LSTM layers to learn joint temporal dynamics across features.

The input shape remains $(40, 23)$, representing 4 seconds of system behavior sampled every 0.1 second. The architecture consists of:

- **Input Layer**: Sequence input with 40 time steps and 23 features per step.
- **Depthwise Conv1D**: 23 filters with `groups=23`, kernel size 3 or 10, and linear activation. Each filter acts on a single feature timeline.
- **MaxPooling1D (optional)**: Applied in some variants to reduce sequence length by half before the LSTM.
- **LSTM Stack**: One or two LSTM layers with 256 or 64 units.
- **Dense Output Layer**: Sigmoid-activated neuron for binary classification.

This architecture focuses on isolating fine-grained patterns within each feature, avoiding interference from irrelevant dimensions. It is particularly well-suited for capturing bursty or sharply localized behavior such as sudden entropy changes or access spikes.

Experiment 1: Kernel Size = 3, 2-LSTM (256–64), With Pooling This is the most balanced depthwise variant. It uses a kernel size of 3, applies max pooling, and stacks two LSTM layers of sizes 256 and 64 respectively.

Parameter	**Value**
Kernel Size	3
Pooling	Yes
LSTM Stack	$256 \rightarrow 64$
Dropout	None
Output Activation	Sigmoid
F1 (Ransomware)	0.8698
Precision	94.0%
Recall	81.2%
Accuracy	82.62%
False Negatives	6546
False Positives	1270

Table 22: Depthwise Conv–LSTM (kernel=3, 2-layer LSTM, with pooling)

Experiment 2: Kernel Size = 3, 2-LSTM (256–256), No Pooling Removing max pooling allows the LSTM layers to observe the full-resolution temporal sequence. This resulted in the highest F1-score among all depthwise variants.

Parameter	**Value**
Kernel Size	3
Pooling	No
LSTM Stack	$256 \rightarrow 256$
Dropout	None
Output Activation	Sigmoid
F1 (Ransomware)	0.8901
Precision	95.0%
Recall	84.3%
Accuracy	84.88%
False Negatives	5133
False Positives	1746

Table 23: Depthwise Conv–LSTM (kernel=3, full LSTM stack, no pooling)

Experiment 3: Kernel = 3, No Pooling, Dropout Between LSTMs To test the effect of regularization, a Dropout layer (rate 0.3) was inserted between the two LSTM layers. This reduced overfitting slightly but also introduced instability in benign prediction.

Parameter	**Value**
Kernel Size	3
Pooling	No
LSTM Stack	256 → 256
Dropout	Between LSTMs (0.3)
Output Activation	Sigmoid
F1 (Ransomware)	0.8721
Precision	93.7%
Recall	81.9%
Accuracy	83.42%
False Negatives	5526
False Positives	1910

Table 24: Depthwise Conv–LSTM with dropout between LSTM layers

Discussion The depthwise approach proved to be highly effective when paired with a deeper LSTM stack and no pooling. It preserves per-feature signals and avoids convolutional interference, but lacks the ability to model inter-feature relationships. Nonetheless, it surpassed the performance of both the baseline LSTM and several unified CNN models in terms of recall and generalization.

Model Variant	**F1**	**Precision**	**Recall**	**Accuracy**	**Observations**
Unified Conv (kernel = 7, no pooling)	**0.9078**	**96.5%**	85.6%	87.33%	Strongest unified architecture; excellent precision with no pooling.
Depthwise Conv (kernel = 3, 256-64, pooling)	0.8698	94.0%	81.2%	82.62%	Most balanced depthwise model; moderate complexity and good recall.
Depthwise Conv (kernel = 3, 2x256, no pooling)	0.8901	95.0%	84.3%	84.88%	Best-performing depthwise setup in terms of recall and generalization.
Unified Conv (softmax output)	0.8963	91.1%	**88.3%**	86.01%	High recall using categorical output; slightly lower precision than sigmoid.
LSTM Only (single layer)	0.8500	93.0%	78.9%	81.22%	Simplified baseline with no convolution; lacks spatial context learning.

Table 25: Detailed comparison of best-performing CNN–LSTM variants evaluated on the family-aware test set. All models use a sequence length of 40 and input dimensionality of 23 features.

9.3 Dual-Channel Hybrid CNN–LSTM Architecture

In an effort to enhance the representational capacity and generalization of our temporal behavior analysis system, we introduced a dual-branch CNN–LSTM architecture. The core idea is to **fuse two complementary temporal pattern extractors**—one focusing on localized trends within each feature independently (depthwise convolution), and the other capturing cross-feature temporal correlations (standard convolution).

Each input sequence of shape (40×23), representing 4 seconds of behavioral telemetry, is **fed in parallel** to two distinct branches:

- **Branch 1: Depthwise CNN $\rightarrow$ LSTM**: Applies a separate 1D convolution to each of the 23 features over time, using grouped convolution with $groups = 23$. This effectively treats each feature as an independent signal, extracting **feature-local temporal filters**. This preserves per-feature semantics and mirrors the design found in the RanSMAP baseline.
- **Branch 2: Unified CNN $\rightarrow$ LSTM**: Uses a traditional Conv1D layer where each filter spans all 23 features at once. This allows for **cross-feature pattern detection**, enabling the model to learn joint activity behaviors (e.g., a burst in CPU + memory + entropy simultaneously).
- **Fusion and Classification**: The two branches produce high-level temporal embeddings (via stacked LSTMs). These are **concatenated** and passed through a fully connected output head (sigmoid for binary classification). The hope is that combining two orthogonal perspectives on the sequence improves discrimination power across unseen ransomware families.

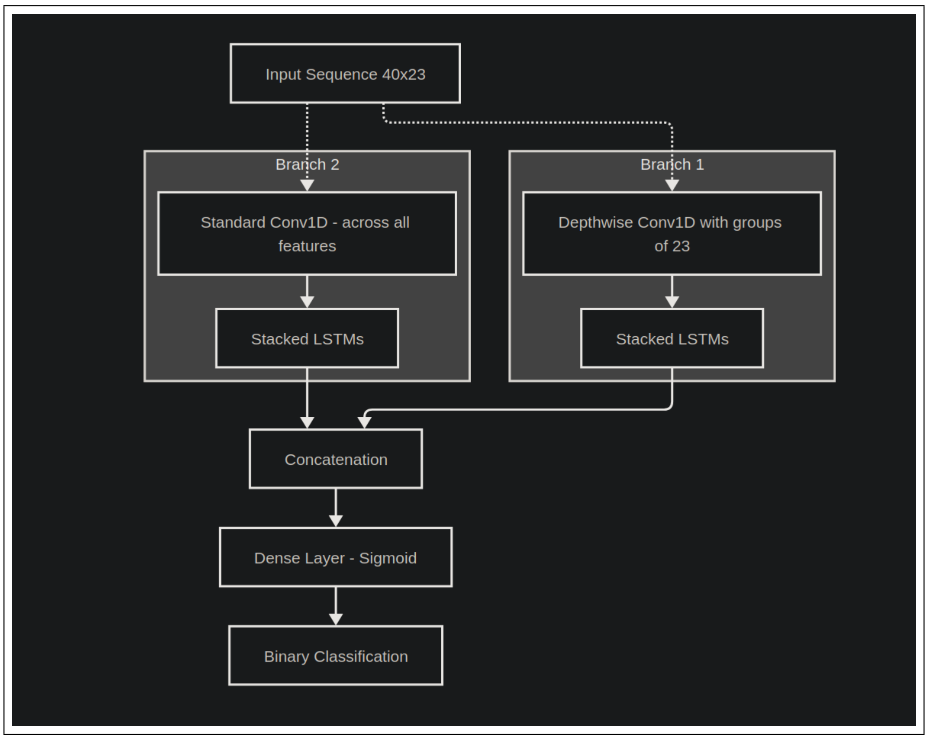


Figure 28: System-level overview of the dual-channel hybrid model. The input is processed in parallel by both depthwise and unified convolutional branches, each followed by stacked LSTMs. Their final representations are fused for classification.

This architecture is motivated by the idea that no single type of temporal filter is sufficient across all ransomware behaviors. Some families exhibit subtle, feature-local shifts (e.g., gradual increase in entropy), while others show bursty multi-feature transitions (e.g., simultaneous I/O and CPU spikes). The dual channel design allows the model to learn both.

The next sections walk through a series of carefully designed experiments, where we explore activation types, pooling strategies, and recurrent stacking depth to understand their impact in this dual setup.

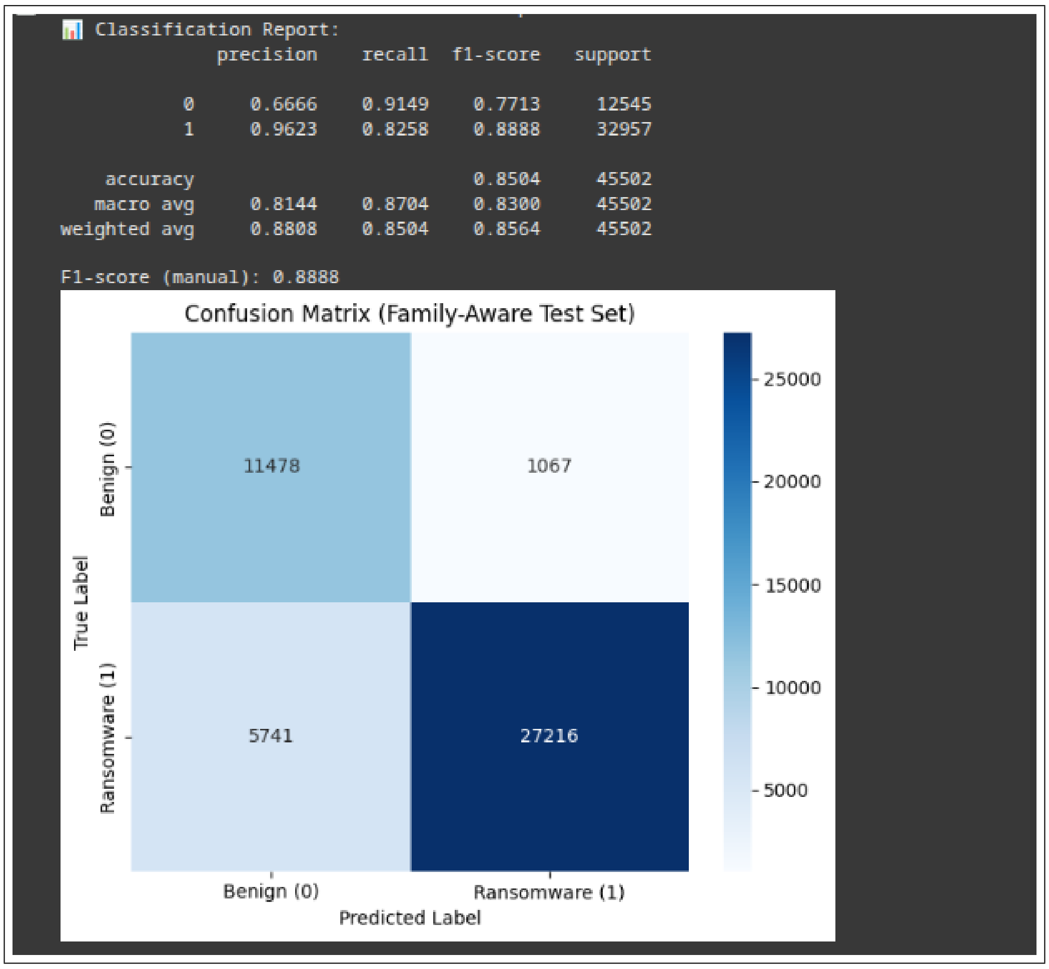
9.3.1 Linear activation + MaxPooling Baseline

Our first experiment using the dual-channel architecture aims to **replicate the structure and behavior of previous best-performing single-path CNN–LSTM models**. We adopt linear convolutional filters in both branches and apply `MaxPooling1D(pool_size=2)` after the convolution to reduce the temporal dimension and emphasize dominant activations.

Configuration:

- **Activation:** Linear in both branches.
- **Kernel Size:** 3 in both branches.
- **Pooling:** MaxPooling (pool_size = 2) applied after convolution.
- **LSTM:** Two stacked LSTM layers with 256 units each in both branches.
- **Fusion:** Concatenation of both LSTM outputs.
- **Output:** Dense(1, activation='sigmoid').

Rationale: The use of linear activation mimics the design from RanSMAP, where convolution acts more like a feature smoother or temporal aggregator rather than a non-linear encoder. MaxPooling is used to reduce sequence length while retaining peak signals, providing a good inductive bias for LSTM to model higher-level transitions.



```
📊 Classification Report:
              precision    recall  f1-score   support

           0     0.6666    0.9149    0.7713     12545
           1     0.9623    0.8258    0.8888     32957

    accuracy                         0.8504     45502
   macro avg     0.8144    0.8704    0.8300     45502
weighted avg     0.8808    0.8504    0.8564     45502

F1-score (manual): 0.8888
```

Figure 29: Confusion Matrix — Linear + MaxPooling Baseline

Metric	**Value**
F1-score (Manual)	**0.8888**
Precision (Ransomware)	96.2%
Recall (Ransomware)	82.6%
Accuracy	85.0%
Macro F1	83.0%
Weighted F1	85.6%

Table 26: Performance Metrics — Linear activation + MaxPooling Baseline

This baseline sets a solid starting point for the dual-branch model. While the recall is somewhat lower than in other configurations, the high precision and stable F1-score demonstrate the value of combining depthwise and unified branches. Later experiments will evaluate how removing pooling or changing activation influences these dynamics.

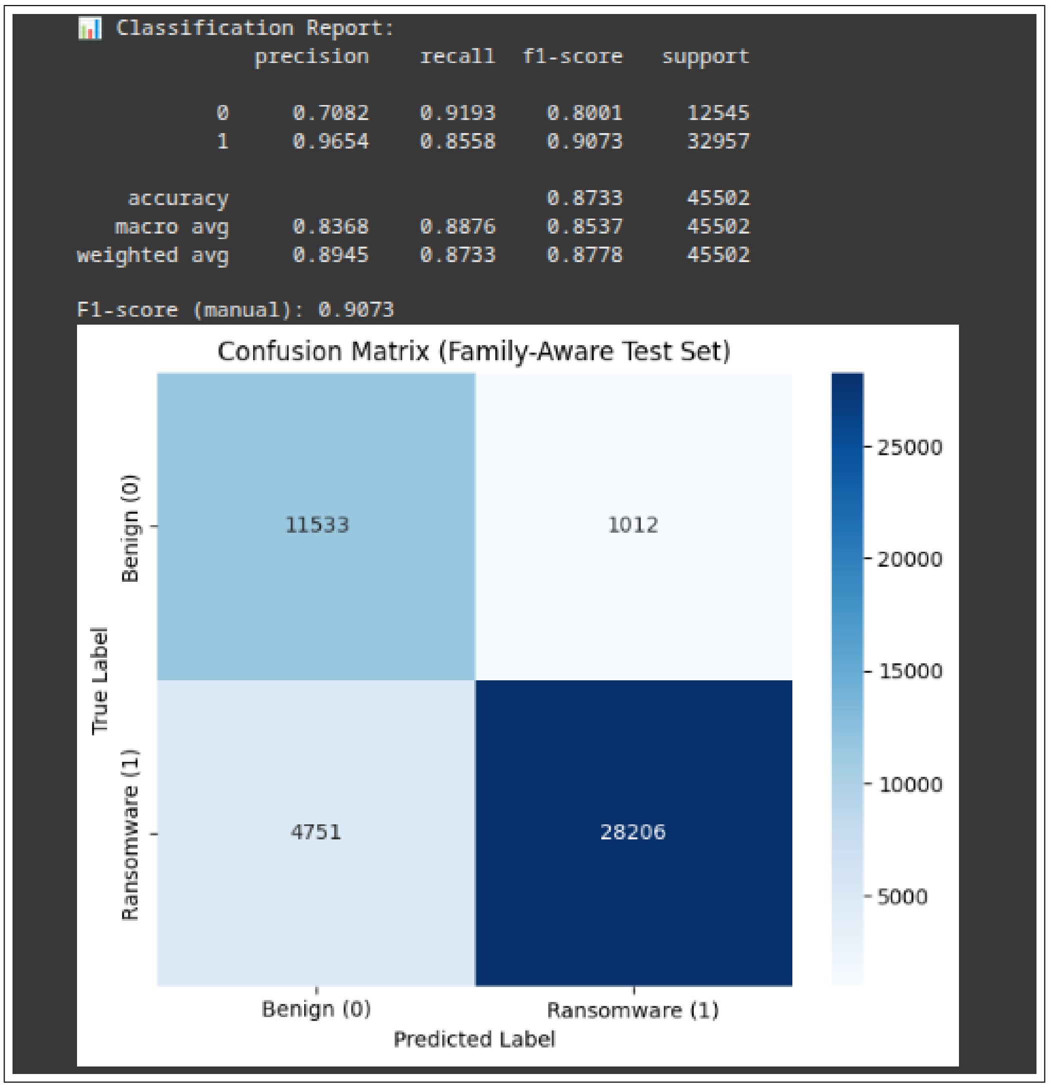
9.3.2 Linear Without Pooling

In this experiment, we analyze the effect of removing temporal downsampling (MaxPooling) while keeping linear activation in both convolution branches. By feeding the full-resolution sequence directly into the LSTM layers, the model retains detailed temporal dynamics that could help capture subtle ransomware behaviors.

Configuration:

- **Activation:** Linear in both branches.
- **Kernel Size:** 3 in both branches.
- **Pooling: None**.
- **LSTM:** Two LSTM layers with 256 units each in both branches.
- **Fusion:** Concatenation of both LSTM outputs.
- **Output:** Dense(1, activation='sigmoid').

Rationale: In some cases, pooling may suppress key sequence-level information that is crucial for modeling slow-developing behaviors typical of ransomware. By skipping pooling, we allow the LSTM to operate over the full 40-timestep sequence, which could provide richer temporal insight — though at a higher computational cost.



```
📊 Classification Report:
              precision    recall  f1-score   support

           0     0.7082    0.9193    0.8001     12545
           1     0.9654    0.8558    0.9073     32957

    accuracy                         0.8733     45502
   macro avg     0.8368    0.8876    0.8537     45502
weighted avg     0.8945    0.8733    0.8778     45502

F1-score (manual): 0.9073
```

Figure 30: Confusion Matrix — Linear Without Pooling

Metric	**Value**
F1-score (Manual)	**0.9073**
Precision (Ransomware)	96.5%
Recall (Ransomware)	85.6%
Accuracy	87.33%
Macro F1	85.3%
Weighted F1	87.78%

Table 27: Performance Metrics — Linear Without Pooling

This experiment outperformed the max-pooling baseline, indicating that the model benefits from full temporal detail when using linear filters. While computationally heavier, it achieves the best performance among linear dual-stream setups — establishing a strong baseline before moving to non-linear activations.

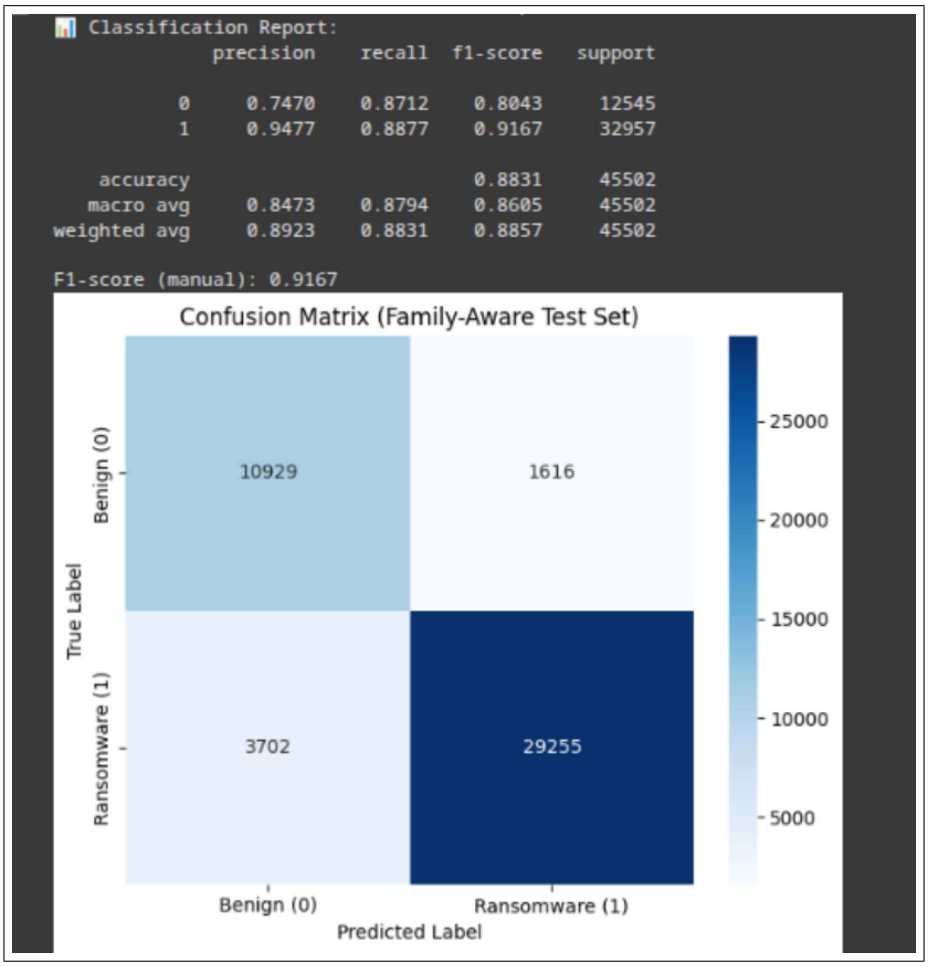
9.3.3 ReLU Activation Without Pooling

This experiment builds on the previous no-pooling architecture by replacing the linear activations in both convolution branches with **ReLU** (Rectified Linear Unit). The goal is to inject non-linearity into the convolutional filtering phase while still allowing the LSTM layers to receive full-resolution sequences.

Configuration:

- **Activation:** ReLU in both convolution branches.
- **Kernel Size:** 3 in both branches.
- **Pooling: None**.
- **LSTM:** Two stacked LSTM layers (256 units each) per branch.
- **Fusion:** Concatenation of final LSTM states from both streams.
- **Output:** Dense(1, activation='sigmoid').

Rationale: While the original depthwise paper (RanSMAP) emphasized preserving linearity for smooth temporal filtering, adding ReLU could allow the model to learn richer local feature abstractions before handing them off to the LSTM. This introduces non-linearity at the expense of possible distortion, which we test empirically.



```
📊 Classification Report:
              precision    recall  f1-score   support

           0     0.7470    0.8712    0.8043     12545
           1     0.9477    0.8877    0.9167     32957

    accuracy                         0.8831     45502
   macro avg     0.8473    0.8794    0.8605     45502
weighted avg     0.8923    0.8831    0.8857     45502

F1-score (manual): 0.9167
```

Figure 31: Confusion Matrix — ReLU Without Pooling

Metric	Value
F1-score (Manual)	**0.9167**
Precision (Ransomware)	94.8%
Recall (Ransomware)	88.7%
Accuracy	88.31%
Macro F1	86.05%
Weighted F1	88.57%

Table 28: Performance Metrics — ReLU Without Pooling

This variant yields a notable performance boost over the linear no-pooling baseline, suggesting that well-placed non-linearities can enhance feature diversity for temporal modeling. It also reaches our highest recall yet, making it a strong candidate when ransomware detection sensitivity is prioritized.

9.3.4 Best Acheived Architecture - Dual-Channel BiLSTM + ReLU Actv

Building on the success of the ReLU-based dual-branch architecture, this final experiment incorporated **Bidirectional LSTM** layers in both the depthwise and unified convolution streams. The motivation was to allow the LSTM layers to access both forward and backward temporal dependencies, potentially improving the understanding of subtle context shifts across the sequence.

This modification aimed to address the hypothesis that some ransomware behaviors exhibit early warning signs that might be missed in strictly forward modeling—thus, enabling the model to observe how the full temporal pattern unfolds from both directions could yield more robust sequence representations.

The architecture retained:

- A shared input of shape $(40, 23)$.
- Two convolution branches:
  - **Branch 1 (Depthwise Conv):** `Conv1D(groups=23, kernel=3, activation=ReLU)`.
  - **Branch 2 (Unified Conv):** `Conv1D(filters=64, kernel=3, activation=ReLU)`.
- **No pooling** to retain full sequence fidelity.
- **Bidirectional LSTMs** (256 units) in each branch.
- Final concatenation of both representations followed by a sigmoid classification head.

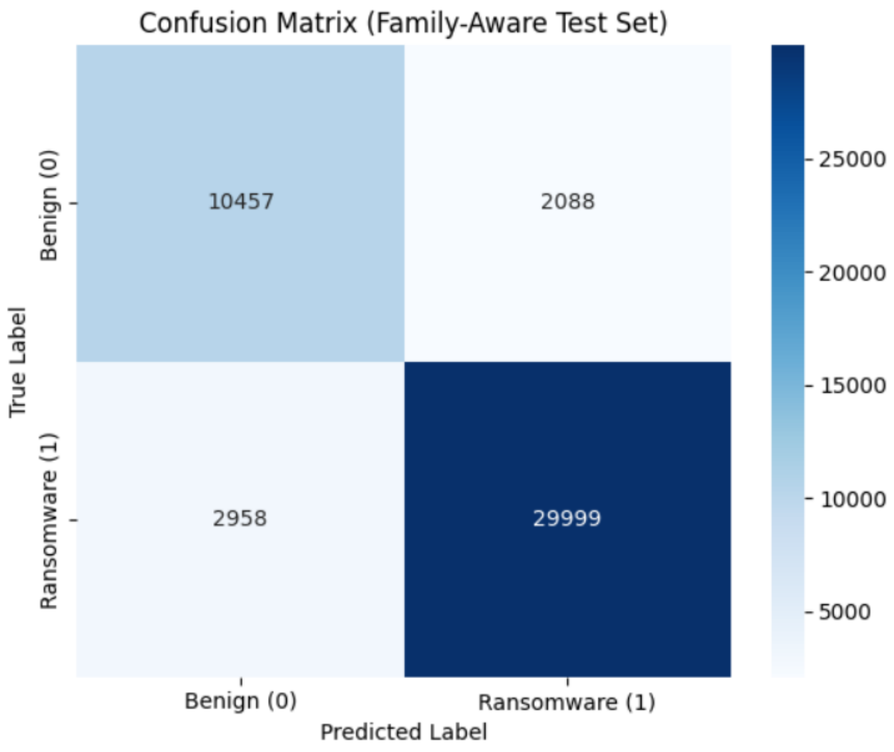
The result was the most performant model across all experiments in terms of F1-score, recall, and accuracy. It significantly outperformed the previously best unified-conv setup and demonstrated superior balance and generalization to unseen ransomware families.

Performance Overview:

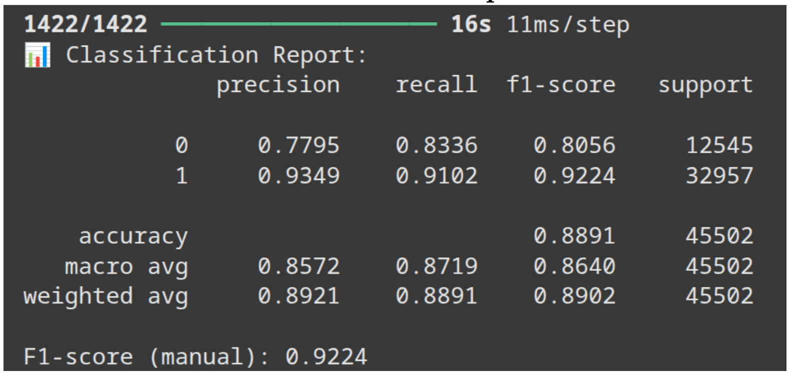
Metric	Value
F1-score (manual)	**0.9224**
Precision (ransomware)	93.49%
Recall (ransomware)	**91.02%**
Accuracy	88.91%
Macro Avg F1	0.8640
Weighted Avg F1	0.8902

Table 29: Dual-Channel BiLSTM Model Performance on Family-Aware Test Set

Confusion Matrix



Classification Report



```
1422/1422 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step
📊 Classification Report:
              precision    recall  f1-score   support

           0     0.7795    0.8336    0.8056     12545
           1     0.9349    0.9102    0.9224     32957

    accuracy                         0.8891     45502
   macro avg     0.8572    0.8719    0.8640     45502
weighted avg     0.8921    0.8891    0.8902     45502

F1-score (manual): 0.9224
```

The dramatic improvement in ransomware recall (91.02%) indicates the model's strong capacity to generalize across behavior patterns, including those not seen during training. This confirms the potential of **feature-parallel bidirectional sequence modeling** as an advanced architecture for behavior-based ransomware detection.

9.3.5 Summary of CNN–LSTM Hybrid Architectures (Family-Aware Split)

The hybrid modeling approach combining Convolutional Neural Networks and Long Short-Term Memory (CNN–LSTM) layers proved to be a transformative design direction for behavior-based ransomware detection. Unlike pure LSTM models that process raw sequential features or CNN-only models that focus on short-term patterns, the hybrids offer a fusion of both: local temporal filtering followed by long-range sequential understanding. This dual strength is particularly effective in malware behavior modeling where both micro-level bursts (e.g., entropy spikes, registry activity) and macro-level trends (e.g., sustained encryption cycles) matter.

All CNN–LSTM architectures were rigorously evaluated using a **family-aware split**, a challenging experimental setup where ransomware families used for training are disjoint from those used in validation and testing. This design simulates real-world *zero-day* ransomware detection, where models must generalize to previously unseen threats. In this high-bar setting, hybrid models consistently demonstrated superior robustness and detection sensitivity.

Among the numerous configurations explored, three evolutionary paths emerged:

1. **Unified Conv + LSTM:** A single convolution layer over the full feature set followed by LSTM layers. These models achieved solid baseline performance and enabled rapid iteration.
2. **Depthwise Conv + LSTM:** Separate convolutions for each feature dimension before temporal modeling. This approach captured individual signal dynamics with precision and proved to be a strong standalone architecture.
3. **Dual-Channel Architectures:** A fusion of unified and depthwise branches processed in parallel. These systems dramatically improved representational richness and outperformed all other models—culminating in the **Bidirectional Dual-Channel model**, which achieved the best generalization to unseen families.

Comparison Table:

Model Variant	F1-score	Precision	Recall	Notes
Dual-Channel BiLSTM (ReLU, no pool)	**0.9224**	93.49%	**91.02%**	Best overall; bidirectional LSTM + dual features
Dual-Channel ReLU (no pool)	0.9167	**94.77%**	88.77%	High precision variant of dual channel
Unified Conv (kernel=7, no pool)	0.9078	96.5%	85.6%	Best among unified-only models
Depthwise Conv (kernel=3, LSTM 256-64)	0.8698	94.0%	81.2%	Balanced and lightweight depthwise design
Single LSTM (no CNN)	0.8704	93.5%	83.1%	Best of standalone LSTM baselines

Table 30: Top-performing CNN–LSTM models evaluated on the family-aware test set

This table reinforces the remarkable gain achieved via **feature-parallel architectures** (dual-branch) and **temporal bidirectionality**. The best hybrid models not only achieved top F1-scores, but also displayed high precision and recall, proving their robustness to diverse unseen ransomware behaviors.

In conclusion, CNN–LSTM hybrids, particularly the dual-branch bidirectional design representing a viable and advanced strategy for real-world ransomware detection systems where behavioral variance and novelty are prevalent.

10 Wavelet Transform Integration (To be continued)

To be continued ...

11 Conclusion

In this research, we set out to improve ransomware detection by enhancing traditional Intrusion Detection Systems (IDS) with powerful AI-driven methods, specifically using Autoencoders, Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) models. Our experiments, conducted on the detailed RanSMAP dataset, sought not only to achieve strong detection accuracy but also to explore how these methods generalize across different ransomware behaviors.

Initially, using a random split—where ransomware families appeared across training and test sets—we achieved near-perfect results (accuracy and F1-scores over 99%). While impressive, these results are most relevant in closed environments where ransomware variants are already familiar. However, in real-world scenarios, IDS must detect zero-day or novel ransomware families, making it crucial to evaluate model robustness through a family-aware split. This approach, where test ransomware families remain completely unseen during training, presents a far more challenging yet realistic evaluation context.

Model	F1-score	Precision	Recall	Accuracy
Random Forest	0.8237	79.23%	85.77%	82.44%
XGBoost	0.8523	84.01%	86.49%	85.66%
MLP	0.6763	62.56%	73.60%	66.30%
CNN (best random split)	0.7206	66.98%	77.98%	71.08%
LSTM (seq=40, best random)	**0.9982**	**99.73%**	**99.91%**	**99.83%**

Table 31: Performance of lightweight models, CNN, and best LSTM on the random split (non-family-aware).

The family-aware evaluation demonstrated clear limitations in simpler models (CNN or Autoencoder alone). Yet, through extensive experimentation—including numerous model architectures, careful hyperparameter tuning, and meticulous preprocessing—we achieved remarkable performance improvements. The best-performing model, our dual-channel bidirectional CNN–LSTM architecture, achieved an F1-score of 92.24% and recall of 91.02% on unseen ransomware families, showcasing excellent generalization and significant practical promise.

Model Architecture	F1-score	Precision	Recall	Accuracy
CNN (single model)	0.7079	68.35%	73.42%	65.95%
Best LSTM (seq=40, 256→64→32)	0.8889	96.12%	82.67%	85.03%
Best Autoencoder (latent classifier calibrated)	0.7990	92.60%	39.80%	72.10%
Unified Conv–LSTM (kernel=7, no pooling)	0.9078	96.50%	85.60%	87.33%
Depthwise Conv–LSTM (kernel=3, LSTM 256–256, no pooling)	0.8901	95.00%	84.30%	84.88%
Dual-channel Bidirectional CNN–LSTM (ReLU, no pooling)	**0.9224**	**93.49%**	**91.02%**	**88.91%**

Table 32: Summary of best-performing models under family-aware evaluation (unseen families in the training set simulating zero-day Ransomware)

The table clearly highlights the substantial performance advantage of the dual-channel bidirectional CNN–LSTM architecture. Compared to simpler models like CNN or autoencoder-based classifiers, the hybrid architectures leveraging temporal context—especially those combining depthwise and unified convolution approaches—demonstrated superior generalization to unseen ransomware families. The calibrated autoencoder provided valuable latent-space insights but struggled in recall, indicating the importance of supervised sequential modeling for behavioral data. These findings reinforce the necessity of richer, sequentially aware model designs for real-world ransomware detection, particularly in challenging family-aware scenarios.

Additionally, we introduced the idea of integrating Discrete Wavelet Transforms (DWT). Although **not yet implemented**, this preprocessing step holds strong potential for further noise reduction and improved detection accuracy. Wavelets could effectively highlight essential behavioral signals, making patterns easier for models to recognize, especially in noisy real-world data.

Practically, our research adds substantial value to existing IDS systems. By deploying these hybrid AI methods alongside traditional signature-based IDS, organizations can greatly enhance their ransomware detection capabilities, particularly against previously unknown threats. Our architecture's ability to generalize robustly across unseen ransomware families makes it highly suitable for practical security contexts.

Academically, this work contributes significant insights into behavioral data processing, model architectures, and evaluation methodologies. Our numerous trials and rigorous exploration of architectural variations yielded clear guidelines on handling sequential behavioral data. Key insights include the necessity of family-aware evaluation, the benefit of dual-channel architectures combining depthwise and unified convolutions, the power of bidirectional LSTMs for capturing comprehensive temporal dynamics, and careful threshold calibration for latent-based models.

In conclusion, this research not only demonstrates strong practical value—significantly enhancing ransomware detection beyond traditional IDS—but also provides valuable methodological and theoretical insights for future cybersecurity research. It offers a clear roadmap for handling complex sequential behavioral data and evaluating IDS robustness against evolving cyber threats.

11.1 Resources

This project leveraged several open-source datasets, reference implementations, and academic studies. Below is a curated list of key resources used throughout the development and evaluation of our models:

- Project Notebooks Repository – All source code and experiments developed during this project.
- RanSMAP Dataset (GitHub) – The primary dataset used for behavioral ransomware modeling.
- RanSMAP Dataset Paper (Elsevier) – A detailed study introducing the RanSMAP dataset and its use in fine-grained behavioral malware detection.
- Building a VAE-Based Network IDS – Demonstrates anomaly detection with autoencoders, including Python code.
- A Transfer Learning and Optimized CNN-Based IDS for IoV – Describes CNN applications with transfer learning in constrained cyber environments.
- Improving Cyber Defense Against Ransomware with GAN+LSTM – Applies adversarial training to improve LSTM classification robustness.
- Effective ML Models (LightGBM/XGBoost) for Ransomware Detection – Benchmarks fast alternatives suitable as fallbacks for deep models.
- Analysis of Network Security Data Using Wavelet Transforms – Examines how wavelet filtering improves IDS signal-to-noise ratio.
- Enhancing Multiclass IDS Using Continuous Wavelet Transform – Supports the value of applying wavelets to complex intrusion patterns.