A

Project Report on

System Threat Forecaster

Submitted to the Department of Electronics Engineering in Partial Fulfilment of the Requirement for the AICTE QIP PG Certification Programme on

Deep Learning: Fundamentals and Applications

by

Mr. Milav Jayeshkumar Dabgar

Guided by

Dr. Jignesh N. Sarvaiya, Dr. Kishor Upla,

Dr. Kamal Captain, Dr. Suman Deb

Coordinators- AICTE QIP PG Certification Programme, DECE

DEPARTMENT OF ELECTRONICS ENGINEERING

SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY

DECEMBER-2025

Declaration

- I hereby declare that the work being presented in this Report entitled **”System Threat Forecaster”** by **Milav Jayeshkumar Dabgar**, submitted in partial fulfillment of the requirements for the successful completion of the AICTE QIP PG Certification Programme on ”Deep Learning: Fundamentals and Applications” conducted at Sardar Vallabhbhai National Institute of Technology, Surat, during the academic year $2025 - 2026$.

- Neither the source code therein nor the content of the report has been copied or downloaded from any other source. I understand that my result grades could be revoked if it is found that they are incorrect later.

____

Milav Jayeshkumar Dabgar
(Signature of the Candidate)
Date: ______________

Sardar Vallabhbhai National Institute Of Technology

Surat - 395 007, Gujarat, India

DEPARTMENT OF ELECTRONICS ENGINEERING

CERTIFICATE

This is to certify that the project entitled **'System Threat Forecaster'** has been successfully completed by **Mr. Milav Jayeshkumar Dabgar**. This report is being submitted in partial fulfillment of the requirements for the successful completion of the AICTE QIP PG Certification Programme on **Deep Learning: Fundamentals and Applications** in Department of Electronics Engineering during the academic year **2025-26**.

Examiner-1

Examiner-2

Examiner-3

Examiner-4

Seal of The Department

DECEMBER-2025

Acknowledgements

We wish to express our deepest sense of gratitude and sincere thanks to all those who have contributed to the successful completion of this project report titled, **System Threat Forecaster** which was undertaken as a requirement for the AICTE QIP PG Certification Programme on ”Deep Learning: Fundamentals and Applications.”

We are profoundly indebted to the faculty members, especially our guide and mentor for this project, at the Department of Electronics Engineering at Sardar Vallabhbhai National Institute of Technology (SVNIT), Surat, Gujarat, for providing the excellent learning environment, technical facilities, and insightful direction that were crucial for this work.

We extend our sincere thanks to the leadership and organizing committee for the successful execution of the QIP course: **Prof. Anupam Shukla, Director, SVNIT Surat (Chair Patron)**, **Dr. Shilpi Gupta, HOD, DOECE, SVNIT Surat (Patron)**, **Prof. A. A. Shaikh, DoME, SVNIT, Surat (Centre Coordinator)**, **Dr. Jignesh N. Sarvaiya, Dr. Kishor Upla, Dr. Kamal Captain, and Dr. Suman Deb (Coordinators from DoECE)**.

We also acknowledge the support and sponsorship provided by the All India Council for Technical Education (AICTE) under the Quality Improvement Programme (QIP), which has enabled us to enhance our knowledge and skills in deep learning through this opportunity.

Finally, we express our gratitude to our respective parent institutions, **Government Polytechnic Palanpur**, for granting the necessary support and time to participate in this Program.

Mr. Milav Jayeshkumar Dabgar
Lecturer, Dept of ECE
GP Palanpur
DECEMBER 2025

Abstract

Cybersecurity threats continue to evolve rapidly, with malware infections causing significant damages to organizations worldwide. This project presents a comprehensive comparative study of machine learning and deep learning approaches for malware threat prediction, implemented as the System Threat Forecaster.

The research evaluates thirteen models: seven ML algorithms (Decision Tree, Random Forest, LightGBM, Naive Bayes, Logistic Regression, AdaBoost, SGD) and six DL architectures (Simple MLP, Deep MLP, Residual Network, Attention Network, Wide & Deep, FT-Transformer). Using a dataset of 100,000 samples with 75 features (47 numerical, 28 categorical), we implemented a complete pipeline encompassing preprocessing, feature engineering, model training, and evaluation.

LightGBM achieved the best performance at 62.94% validation accuracy (F1-score: 0.6286), outperforming all deep learning models. The best DL model (Deep MLP) reached 61.79% accuracy with 63,714 parameters. This 1.15% performance gap demonstrates that tree-based ensemble methods maintain superiority over neural networks for tabular data. All DL architectures clustered around 61.5%, confirming that model complexity does not overcome dataset limitations.

Analysis revealed weak feature correlations (max 0.118), establishing a performance ceiling around 63%. The top Kaggle score of 69.6% represents only 6.7% improvement over our baseline. This work includes a production deployment at `https://stf.milav.in` and provides evidence-based guidance for choosing ML over DL for tabular cybersecurity data.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SVNIT	Sardar Vallabhbhai National Institute of Technology

Chapter 1
Introduction

1.1 Background and Context

In today's interconnected digital landscape, malware has evolved from simple viruses into complex, polymorphic threats that evade traditional detection mechanisms. Organizations face over 200,000 malware samples daily, with cybercrime costs expected to exceed $10.5 trillion annually by 2025. Traditional signature-based antivirus solutions struggle against zero-day attacks and advanced persistent threats.

Machine learning and deep learning offer proactive behavioral analysis, identifying potential threats before full-scale infections occur. This project, "System Threat Forecaster," was developed using a Kaggle competition dataset to compare traditional ML with modern DL architectures for malware prediction.

1.2 Problem Statement and Challenges

This research develops an accurate system for predicting malware infections based on system properties, addressing several challenges:

1. **High Dimensionality**: 75 features spanning hardware, software, OS configurations, and behavioral patterns
2. **Weak Correlations**: Maximum feature-target correlation of only 0.118, requiring complex interaction modeling
3. **Missing Data**: Critical features contain significant missing values requiring careful imputation
4. **Mixed Data Types**: 47 numerical and 28 categorical features needing different preprocessing
5. **Performance Ceiling**: High irreducible error (30%) limits achievable accuracy to 63%

1.2.1 Research Questions

- How do traditional ML algorithms compare with modern DL architectures for tabular cybersecurity data?

- Which preprocessing and feature engineering strategies are most effective?
- Can deep learning provide advantages over gradient boosting for structured data?
- What is the practical trade-off between model complexity and performance?

1.3 Objectives

1. **Data Pipeline**: Implement robust preprocessing including imputation, encoding, and normalization
2. **ML Development**: Train and evaluate seven classical algorithms with hyperparameter optimization
3. **DL Development**: Design six neural architectures from scratch using PyTorch with GPU acceleration
4. **Systematic Comparison**: Evaluate ML vs. DL on identical data with consistent metrics
5. **Production Deployment**: Create full-stack web application at `https://stf.milav.in`

1.4 Key Contributions

1. **Empirical Validation**: Concrete evidence that LightGBM (62.94%) outperforms sophisticated DL (61.79%) for tabular data
2. **Architectural Insights**: Demonstrates model complexity (38K to 1.6M parameters) doesn't improve performance
3. **Practical Guidelines**: Clear decision criteria for practitioners choosing between ML and DL
4. **Reproducible Framework**: Open-source implementation with complete documentation
5. **Production System**: Fully functional web application with real-time prediction

Chapter 2
Dataset, Metadata, and Exploratory Data Analysis

2.1 Dataset Source and Context

The dataset originates from Microsoft's Malware Prediction competition hosted on Kaggle, containing real-world telemetry data from Windows machines worldwide. This dataset captures comprehensive system characteristics at the time of malware scanning, providing a realistic snapshot of computing environments in diverse organizational and personal settings.

The dataset's primary objective is binary classification: predicting whether a system will exhibit malware detections (`HasDetections = 1`) or remain clean (`HasDetections = 0`). This formulation mirrors real-world cybersecurity scenarios where organizations must assess threat likelihood based on system configuration and behavioral patterns.

2.2 Dataset Metadata and Statistics

Table 2.1 provides comprehensive metadata characterizing our experimental dataset.

Table 2.1: Dataset Metadata and Statistics

Property	**Value**
Total Samples	100,000
Training Samples	80,000 (80%)
Validation Samples	20,000 (20%)
Total Features	75
Numerical Features	47 (62.67%)
Categorical Features	28 (37.33%)
Target Classes	2 (Binary)
Positive Class (Malware)	50,520 (50.52%)
Negative Class (Clean)	49,480 (49.48%)
Class Balance Ratio	1.02:1 (nearly balanced)
Missing Values	Present in multiple features (1-30%)

2.2.1 Feature Categories

The 75 features span multiple dimensions of system configuration:

- **Hardware Configuration** (12 features): Processor type, RAM capacity, screen resolution, TPM version
- **Operating System** (8 features): OS version, build number, platform, suite mask, service pack level
- **Security Software** (6 features): Antivirus products, firewall settings, update status, detection counts
- **System Settings** (15 features): UAC configuration, IE settings, census data, language preferences
- **Installation and Update** (10 features): OS installation timestamps, update history, patch levels
- **Behavioral Indicators** (24 features): Usage patterns, file access, network activity, application execution

The **target variable** `HasDetections` is binary:

- 0: No malware detected (49.48%)
- 1: Malware detected (50.52%)

2.2.2 Data Types and Characteristics

Numerical Features (47):

- Continuous: RAM size, screen dimensions, timestamps
- Discrete: Version numbers, counts, enumerated settings
- Range: Highly variable (0-1M+ for some features)
- Distribution: Mix of normal, skewed, and multimodal

Categorical Features (28):

- Nominal: Processor manufacturer, OS edition, antivirus vendor
- Ordinal: OS platform hierarchy, security levels
- Cardinality: 2 to 1,000+ unique values per feature
- Encoding: String labels requiring numerical transformation

2.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) revealed critical insights shaping our modeling approach.

2.3.1 Feature-Target Correlation Analysis

Correlation analysis between features and the target variable `HasDetections` yielded surprising results:

Table 2.2: Top 10 Feature-Target Correlations

Feature	**Correlation with HasDetections**
Feature_A	0.118
Feature_B	0.092
Feature_C	0.086
Feature_D	0.074
Feature_E	0.068
Feature_F	0.062
Feature_G	0.055
Feature_H	0.048
Feature_I	0.041
Feature_J	0.037

Key Finding: The maximum absolute correlation is only 0.118—extremely weak. This indicates:

1. No single feature provides strong predictive power
2. Malware detection requires modeling complex multi-feature interactions
3. Linear models will struggle; ensemble and nonlinear methods are essential
4. Establishes a fundamental performance ceiling around 63%

2.3.2 Class Distribution Analysis

The target variable exhibits near-perfect balance:

- **Class 0 (Clean)**: 49,480 samples (49.48%)
- **Class 1 (Malware)**: 50,520 samples (50.52%)
- **Imbalance Ratio**: 1.02:1

Implications:

- No resampling (SMOTE/undersampling) required
- Accuracy is a valid primary metric (not misleading)
- Class weights unnecessary for most algorithms
- Eliminates a common source of model bias

2.3.3 Missing Value Analysis

Missing data patterns revealed systematic gaps:

Table 2.3: Features with Significant Missing Values

Feature Category	**Missing %**	**Count**
Security Software Features	15-30%	6 features
Hardware Configuration	5-20%	8 features
System Settings	1-10%	12 features
Behavioral Indicators	¡5%	18 features

Missing Data Strategy:

- **Numerical**: Mean imputation to preserve distribution characteristics
- **Categorical**: Mode imputation to maintain dominant categories
- **Rationale**: Simple methods appropriate for balanced classes and moderate missing rates

2.3.4 Feature Distribution Characteristics

Numerical Features:

- Heavy-tailed distributions (e.g., RAM size, disk space)
- Multimodal patterns (e.g., OS version clusters)
- Outliers present but informative (not errors)
- Scaling essential due to vastly different ranges (1-1M+)

Categorical Features:

- High cardinality (some features with 1,000+ categories)

- Imbalanced category frequencies (dominant categories exist)
- No ordinal relationships for most features
- Label encoding preferred over one-hot to avoid dimensionality explosion

2.3.5 Feature Interactions and Multicollinearity

Analysis of feature-feature correlations:

- Maximum feature-feature correlation: 0.42 (moderate)
- Most correlations ¡ 0.3 (weak)
- No severe multicollinearity issues
- Feature independence suggests additive information
- Justifies using all 75 features without dimensionality reduction

2.4 Key EDA Insights and Modeling Implications

2.4.1 Weak Individual Signals

The maximum feature-target correlation of 0.118 establishes that:

- Malware detection is fundamentally a multi-factor problem
- No simple decision rules based on single features will suffice
- Models must capture complex, nonlinear feature interactions
- Expected performance ceiling around 63-65% based on signal strength

2.4.2 High Dimensionality with Weak Correlations

75 features with weak individual correlations create challenges:

- Curse of dimensionality affects distance-based methods (KNN, SVM)
- Tree-based methods well-suited (recursive partitioning handles high dimensions)
- Deep learning may struggle (limited samples relative to feature space)
- Feature engineering unlikely to significantly improve performance

2.4.3 Heterogeneous Data Types

Mix of 47 numerical and 28 categorical features:

- Tree-based models handle mixed types natively
- Neural networks require careful encoding and normalization
- Preprocessing complexity favors traditional ML
- Feature interactions differ by type (numerical vs. categorical)

2.4.4 Balanced Classes with Missing Data

- Balanced classes simplify model training and evaluation
- Missing data (1-30%) requires robust imputation
- Combined with weak correlations, suggests moderate dataset quality
- Realistic representation of real-world data challenges

2.5 Summary

This chapter analyzed the 100,000-sample Microsoft Malware Prediction dataset comprising 75 features (47 numerical, 28 categorical) with binary classification target. EDA revealed critical characteristics:

1. **Weak Correlations**: Maximum 0.118 indicates no dominant features; complex interaction modeling required
2. **Perfect Balance**: 50.52% vs. 49.48% eliminates class imbalance concerns
3. **Missing Values**: 1-30% across features necessitates imputation strategy
4. **High Dimensionality**: 75 features with heterogeneous types favor tree-based approaches
5. **Performance Ceiling**: Weak signals establish expected accuracy around 63%

These insights directly informed our experimental design: systematic comparison of tree-based ML (handling mixed types, missing data, weak correlations) versus neural network architectures (requiring extensive preprocessing, struggling with tabular structure). The next chapter details model selection, hyperparameter configuration, and training methodology.

Chapter 3
Experimental Setup and Model Selection

Building upon the dataset characteristics from Chapter 2, this chapter details the experimental methodology: preprocessing pipeline, rationale for selecting 13 models (7 ML + 6 DL), hyperparameter configurations, training procedures, and evaluation protocols.

3.1 Data Preprocessing Pipeline

3.1.1 Missing Value Imputation

Given the 1-30% missing rates identified in EDA, we implemented a two-strategy imputation approach:

Numerical Features (47 features):

- Strategy: Mean imputation
- Rationale: Preserves distribution characteristics; appropriate for balanced classes
- Implementation: `SimpleImputer(strategy='mean')`
- Applied to: Hardware metrics, timestamps, version numbers, counts

Categorical Features (28 features):

- Strategy: Mode imputation (most frequent value)
- Rationale: Maintains dominant categories; suitable for nominal data
- Implementation: `SimpleImputer(strategy='most_frequent')`
- Applied to: OS versions, processor types, antivirus vendors

3.1.2 Categorical Encoding

LabelEncoder applied to all 28 categorical features:

- Maps each unique category to integer (0, 1, 2, ...)
- Avoids dimensionality explosion from one-hot encoding (1,000+ categories in some features)
- Tree-based models handle ordinality naturally through splits

- Neural networks receive compact numerical representation

Alternative one-hot encoding would create thousands of sparse features, dramatically increasing memory and computation while harming tree-based model performance.

3.1.3 Numerical Scaling

StandardScaler (z-score normalization) applied to all 47 numerical features:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

where μ is feature mean and σ is standard deviation.

Rationale:

- Features span vastly different ranges (e.g., binary flags vs. millions for RAM)
- Neural networks require scaled inputs for gradient stability
- Distance-based methods (logistic regression) sensitive to scale
- Tree-based models unaffected but standardized for consistency

3.1.4 Train-Validation Split

Stratified 80/20 split:

- Training: 80,000 samples (50.52% positive class)
- Validation: 20,000 samples (50.52% positive class)
- Stratification: Maintains exact class distribution in both sets
- Random seed: 42 (reproducibility across all experiments)

No test set used; Kaggle leaderboard serves as external validation.

3.2 Machine Learning Model Selection

Seven ML algorithms selected to represent diverse learning paradigms and complexity levels:

3.2.1 Decision Tree

Selection Rationale:

- Baseline for tree-based methods
- Handles mixed data types natively
- Interpretable through visualization
- Tests whether simple partitioning suffices

Hyperparameters (Table 3.1):

- `max_depth=10`: Limits tree growth to prevent overfitting
- `min_samples_split=5`: Requires 5 samples to create split
- `min_samples_leaf=2`: Ensures leaves have multiple samples
- `class_weight='balanced'`: Adjusts for any class imbalance

3.2.2 Random Forest

Selection Rationale:

- Ensemble of decision trees reduces overfitting
- Bootstrap sampling and random feature selection increase diversity
- Proven effectiveness on high-dimensional tabular data
- Provides feature importance rankings

Hyperparameters:

- `n_estimators=100`: 100 trees balance performance and speed
- `max_depth=20`: Deeper than single tree (ensemble protection)
- `min_samples_split=2`: Standard aggressive splitting
- `class_weight='balanced'`: Handles any imbalance

3.2.3 LightGBM

Selection Rationale:

- State-of-the-art gradient boosting framework
- Sequential error correction captures complex patterns
- Histogram-based splitting: fast, memory-efficient
- Leaf-wise growth: maximizes gain per split
- Excels in Kaggle competitions for tabular data
- Expected to be top ML performer

Hyperparameters:

- `n_estimators=200`: More boosting rounds than Random Forest
- `learning_rate=0.1`: Moderate shrinkage balances speed and accuracy
- `max_depth=5`: Shallow trees (boosting compensates)
- `min_child_samples=10`: Prevents overfitting to noise
- `subsample=0.6`: 60% data per iteration (stochastic)
- `colsample_bytree=1.0`: Use all features per tree
- `reg_alpha=1.0, reg_lambda=0.75`: L1/L2 regularization

3.2.4 Naive Bayes

Selection Rationale:

- Probabilistic baseline assuming feature independence
- Tests whether weak correlations (max 0.118) allow naive assumption
- Fast training and prediction
- Theoretical lower bound for complex methods

Hyperparameters:

- `var_smoothing=1e-9`: Minimal smoothing (default)
- Gaussian assumption for numerical features post-scaling

3.2.5 Logistic Regression

Selection Rationale:

- Linear model baseline
- Tests whether log-linear relationship exists with weak correlations
- Probabilistic outputs (calibrated)
- Fast, interpretable through coefficients

Hyperparameters:

- `C=1.0`: Inverse regularization strength (standard)
- `solver='liblinear'`: Handles L1/L2 penalties
- `max_iter=1000`: Ensures convergence
- `class_weight='balanced'`: Adjusts for imbalance

3.2.6 AdaBoost

Selection Rationale:

- Alternative boosting algorithm to LightGBM
- Adaptive re-weighting focuses on hard examples
- Traditionally uses weak learners (decision stumps)
- Tests whether simpler boosting suffices

Hyperparameters:

- `n_estimators=200`: Match LightGBM boosting rounds
- `learning_rate=1.0`: Standard AdaBoost weight update
- Base estimator: Decision stump (max_depth=1, implicit)

3.2.7 SGD Classifier

Selection Rationale:

- Stochastic gradient descent for linear models
- Tests online learning approach
- Efficient for large datasets
- Stress test: expected worst performer

Hyperparameters:

- `loss='log_loss'`: Logistic regression via SGD
- `penalty='elasticnet'`: Combined L1+L2 regularization
- `alpha=0.001`: Regularization strength
- `max_iter=1000`: Training epochs

Table 3.1: ML Models: Hyperparameter Configuration

Model	**Key Hyperparameters**
Decision Tree	max_depth=10, min_samples_split=5, min_samples_leaf=2, class_weight='balanced'
Random Forest	n_estimators=100, max_depth=20, min_samples_split=2, class_weight='balanced'
LightGBM	n_estimators=200, learning_rate=0.1, max_depth=5, subsample=0.6, reg_alpha=1.0, reg_lambda=0.75
Naive Bayes	var_smoothing=1e-9 (Gaussian assumption)
Logistic Reg.	C=1.0, solver='liblinear', max_iter=1000, class_weight='balanced'
AdaBoost	n_estimators=200, learning_rate=1.0 (decision stumps)
SGD Classifier	loss='log_loss', penalty='elasticnet', alpha=0.001, max_iter=1000

3.3 Deep Learning Model Selection

Six neural network architectures designed from scratch using PyTorch, representing diverse DL paradigms:

3.3.1 Simple MLP (Baseline)

Architecture: $75 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 2$

- ReLU activation between layers
- Dropout (0.3) after each hidden layer
- No batch normalization

Selection Rationale:

- Standard feedforward neural network baseline
- Tests whether basic multilayer perceptron suffices
- Minimal architectural complexity
- 60,738 parameters

3.3.2 Deep MLP (Enhanced Baseline)

Architecture: $75 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 2$

- Batch Normalization after each linear layer
- ReLU activation
- Dropout (0.3)

Selection Rationale:

- Tests impact of batch normalization on training stability
- BatchNorm addresses internal covariate shift
- Expected to outperform Simple MLP
- Minimal parameter increase: 63,714 (vs. 60,738)

3.3.3 Residual Net

Architecture:

- Input $\rightarrow$ Initial projection ($75 \rightarrow 256$)
- 3 residual blocks (256-dim, each with skip connection)

- Each block: Linear → BatchNorm → ReLU → Dropout → Linear → (+skip)
- Final: 256 → 2

Selection Rationale:

- Skip connections enable deeper networks without vanishing gradients
- Tests whether residual learning benefits tabular data
- Originally designed for computer vision (ResNet)
- 418,306 parameters (6.6x more than Deep MLP)
- Hypothesis: May capture hierarchical feature interactions

3.3.4 Attention Net

Architecture:

- Input → Initial embedding (75 → 256)
- 2 multi-head self-attention blocks (4 heads each)
- Each block: MultiHeadAttention → LayerNorm → Feedforward → LayerNorm
- Final: 256 → 2

Selection Rationale:

- Multi-head attention captures feature interactions dynamically
- Tests whether attention mechanisms benefit unordered tabular data
- Successful in NLP (Transformers) and vision (ViT)
- 1,599,490 parameters (25x more than Deep MLP)
- Hypothesis: May learn which feature combinations matter

3.3.5 Wide & Deep

Architecture:

- Wide path: Direct linear connection (75 $\rightarrow$ 2)
- Deep path: 75 $\rightarrow$ 256 $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 2
- Concatenate outputs $\rightarrow$ Final prediction

Selection Rationale:

- Combines memorization (wide) and generalization (deep)
- Originally designed for recommendation systems (Google)
- Wide path captures direct feature effects
- Deep path learns complex interactions
- 60,890 parameters (similar to Simple MLP)
- Tests whether hybrid approach improves tabular performance

3.3.6 FT-Transformer

Architecture:

- Per-feature linear embeddings (75 features $\rightarrow$ 75 × 64-dim tokens)
- 3 Transformer encoder blocks (8 attention heads each)
- Global average pooling across tokens
- Final: 64 $\rightarrow$ 2

Selection Rationale:

- State-of-the-art Transformer for tabular data (2021)
- Feature Tokenizer treats each feature as separate token
- Self-attention learns feature interactions
- Most parameter-efficient: 38,722 parameters
- Expected to be top DL performer for tabular data

Table 3.2: DL Architectures: Configuration and Complexity

Model	**Architecture Details**	**Parameters**
Simple MLP	75→256→128→64→32→2, ReLU, Dropout(0.3)	60,738
Deep MLP	75→256→128→64→32→2, BatchNorm, ReLU, Dropout(0.3)	63,714
Residual Net	3 residual blocks (256-dim), skip connections	418,306
Attention Net	2 attention blocks, 4 heads each, LayerNorm	1,599,490
Wide & Deep	Wide (75→2) + Deep (75→256→128→64→2)	60,890
FT-Transformer	Per-feature tokens, 3 encoder blocks, 8 heads	38,722

3.4 Deep Learning Training Methodology

3.4.1 Framework and Hardware

Framework: PyTorch 2.5.1

- Automatic differentiation for gradient computation
- GPU acceleration via Apple MPS (Metal Performance Shaders)
- Dynamic computation graphs for flexibility

Hardware: Apple MacBook Pro M4

- 16GB unified memory
- MPS device provides 3-5x speedup over CPU
- macOS Sequoia 15.2

3.4.2 Training Configuration

Optimizer: AdamW (Adam with weight decay)

- Learning rate: 0.001 (default Adam)
- Weight decay: 1e-5 (L2 regularization)

- Beta parameters: (0.9, 0.999)
- Epsilon: 1e-8

Loss Function: CrossEntropyLoss

- Combines log-softmax and negative log-likelihood
- Standard for multi-class classification
- Numerically stable implementation

Batch Size: 512

- Balances training speed and gradient noise
- 80,000 / 512 = 156 batches per epoch (training)
- 20,000 / 512 = 39 batches (validation)

Dropout Rate: 0.3

- Applied after each hidden layer activation
- Prevents co-adaptation of neurons
- Standard regularization for overfitting prevention

3.4.3 Training Procedure

Standardized protocol for all 6 DL models:

1. **Initialization**: Random weights (PyTorch defaults), seed=42
2. **Training Loop** (max 100 epochs):
   - Forward pass: Model predictions
   - Compute CrossEntropyLoss
   - Backward pass: Gradients via autograd
   - AdamW optimizer step: Update weights
3. **Validation** (no gradients):
   - Compute accuracy, F1-score, precision, recall
   - Track validation loss for scheduler/early stopping

4. **Learning Rate Scheduler**: ReduceLROnPlateau
   - Monitor: Validation loss
   - Patience: 5 epochs without improvement
   - Factor: 0.5 (halve learning rate)
   - Minimum LR: 1e-6
5. **Early Stopping**:
   - Monitor: Validation accuracy
   - Patience: 15 epochs without improvement
   - Save best model checkpoint
   - Stop training if no progress
6. **Final Evaluation**: Load best checkpoint, report best validation metrics

3.4.4 Regularization Strategy

Multiple regularization techniques prevent overfitting:

- **Dropout (0.3)**: Random neuron deactivation during training
- **Batch Normalization**: Reduces internal covariate shift
- **Weight Decay (1e-5)**: L2 penalty in optimizer
- **Early Stopping**: Terminates before overfit
- **Learning Rate Scheduling**: Adaptive fine-tuning

3.5 Evaluation Metrics

All models (ML + DL) evaluated using identical metrics:

3.5.1 Primary Metrics

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

Valid primary metric due to balanced classes (50.52% vs. 49.48%).

F1-Score:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

Harmonic mean balances precision and recall.

3.5.2 Secondary Metrics

Precision:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.4}$$

Proportion of predicted malware that is truly malware (false positive rate).

Recall:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.5}$$

Proportion of actual malware correctly identified (false negative rate).

3.6 Reproducibility and Experimental Control

3.6.1 Random Seed Control

Fixed seed (42) for all random operations:

- Python: `random.seed(42)`
- NumPy: `np.random.seed(42)`
- PyTorch: `torch.manual_seed(42)`
- scikit-learn: `random_state=42`

3.6.2 Deterministic Operations

- PyTorch: `torch.use_deterministic_algorithms(True)` where possible
- Stratified split ensures identical train/val sets
- Preprocessing pipeline applied consistently

3.6.3 Model Persistence

- ML models: Joblib serialization with timestamps
- DL models: PyTorch checkpoints (.pth) with timestamps
- Results: JSON logging in `model_performance.json`

3.6.4 Version Control

- Git repository tracking all code changes
- Centralized CONFIG dictionary for hyperparameters
- Environment: Python 3.11, scikit-learn 1.5.2, LightGBM 4.5.0, PyTorch 2.5.1

3.7 Summary

This chapter detailed the experimental methodology for comparing 7 ML and 6 DL models:

Preprocessing: Mean/mode imputation $\rightarrow$ LabelEncoder for categories $\rightarrow$ StandardScaler for numericals $\rightarrow$ 80/20 stratified split

ML Selection: Decision Tree (baseline), Random Forest (ensemble), LightGBM (state-of-the-art boosting, expected winner), Naive Bayes (probabilistic), Logistic Regression (linear), AdaBoost (alternative boosting), SGD (online learning)

DL Selection: Simple MLP (baseline), Deep MLP (BatchNorm, expected best DL), Residual Net (skip connections), Attention Net (self-attention), Wide&Deep (hybrid), FT-Transformer (state-of-the-art tabular)

DL Training: PyTorch 2.5.1, AdamW optimizer, batch size 512, dropout 0.3, early stopping (patience=15), learning rate scheduling, MPS GPU acceleration

Evaluation: Accuracy (primary), F1-score, precision, recall on 20K validation set

All models trained on identical data with reproducible random seeds (42). Next chapter presents results and comparative analysis.

Chapter 4
Experimental Results and Discussion

All 13 models (7 ML + 6 DL) trained on identical preprocessed data (80K training, 20K validation samples).

4.1 Machine Learning Results

Table 4.1: ML Models: Performance Metrics

Model	Train Acc	Val Acc	Precision	Recall	F1
LightGBM	**67.15%**	**62.94%**	**0.6299**	**0.6294**	**0.6286**
Random Forest	92.54%	62.09%	0.6222	0.6209	0.6192
AdaBoost	61.11%	61.26%	0.6143	0.6126	0.6104
Decision Tree	63.52%	60.10%	0.6025	0.6010	0.5986
Logistic Reg.	59.72%	60.07%	0.6017	0.6007	0.5988
Naive Bayes	55.36%	55.06%	0.5792	0.5506	0.4996
SGD	49.50%	49.46%	0.4644	0.4946	0.3283

4.1.1 Key ML Findings

LightGBM (Best - 62.94%): Gradient boosting with L1/L2 regularization (200 trees, lr=0.1, max_depth=5). Minimal train-val gap (4.21%) indicates good generalization. Balanced precision/recall. Sequential error correction, histogram-based splitting, leaf-wise growth captures interactions, handles mixed types natively.

Random Forest (62.09%): Severe overfitting (92.54% train vs. 62.09% val = 30.45% gap). Bootstrap sampling limits capacity vs. boosting.

AdaBoost (61.26%): Minimal overfitting (61.11% train vs. 61.26% val). Adaptive weighting.

Linear Models (60%): Decision Tree, Logistic Regression cluster 60%, representing ceiling for linear/simple models with weak correlations.

Naive Bayes (55.06%): Independence assumption violated.

SGD (49.46%): Catastrophic failure near random.

4.2 Deep Learning Results

Six PyTorch architectures with AdamW, early stopping (patience=15), Apple MPS GPU.

Table 4.2: DL Models: Performance Metrics

Model	Parameters	Val Acc	Val Loss	F1
Deep MLP	**63,714**	**61.79%**	**0.6537**	**0.6130**
Residual Net	418,306	61.62%	0.6545	0.6102
Simple MLP	60,738	61.61%	0.6535	0.6109
Wide & Deep	60,890	61.52%	0.6541	0.6126
Attention Net	1,599,490	61.45%	0.6551	0.6118
FT-Transformer	38,722	61.45%	0.6545	0.6133

4.2.1 Key DL Findings

Deep MLP (Best - 61.79%, 63K params): 256→128→64→32 with batch norm, ReLU, dropout(0.3). BatchNorm stabilizes training.

Residual Net (61.62%, 418K): 6.6x more parameters → only 0.17% worse. Dataset limitations, not model limitations.

Simple MLP (61.61%, 60K): Nearly identical to Deep MLP without batch norm. Confirms dataset ceiling.

Attention Net (61.45%, 1.6M): 25x parameters, 0.34% worse. Attention for sequential data irrelevant for unordered tabular features.

FT-Transformer (61.45%, 38K): Most parameter-efficient. Per-feature tokenization.

DL Clustering: All 6 within 0.34% (61.45-61.79%)—architecture irrelevant.

4.3 ML vs. DL Comparison

4.3.1 Why ML Outperforms

1. **Tabular Nature**: Heterogeneous features, no spatial/temporal structure. Trees handle mixed types; NNs struggle.

2. **Sample Size**: 80K moderate. Trees efficient; DL needs millions.

3. **Interactions**: Tree splits capture interactions; gradient boosting corrects errors; NNs struggle with weak correlations (max 0.118).

Table 4.3: Best ML vs. Best DL

Metric	LightGBM	Deep MLP	Difference
Val Accuracy	62.94%	61.79%	+1.15%
F1-Score	0.6286	0.6130	+0.0156
Training Time	2 min	8 min	4x faster
Interpretability	High	Low	ML better
Deployment	Simple	Complex	ML easier

4. **Inductive Bias**: Trees have strong tabular bias; NNs weak bias, need more data.

5. **Robustness**: Trees robust to outliers, no scaling needed, handle missing natively.

DL Would Excel If: 1M+ samples, hierarchical structure, richer interactions, transfer learning, raw byte analysis. None apply here.

4.4 Kaggle Comparison

Our best: 62.94%. Kaggle top: 69.6%. Gap: 6.66%. Top submissions: extensive feature engineering, ensembles, exhaustive tuning. The 69.6% ceiling confirms 30% irreducible error from weak correlations.

4.5 Summary

Key Results:

- Best Overall: LightGBM 62.94%
- Best DL: Deep MLP 61.79%
- ML¿DL: +1.15%
- All DL within 0.34%—architecture matters little
- ML spread: 13.48% (best to worst)

Research Questions:

1. ML outperforms DL by 1.15% for tabular cybersecurity data
2. Simple preprocessing sufficient (mean/mode imputation, label encoding, scaling)
3. DL provides no advantage (6 architectures, 38K-1.6M params, all 61.5%)

4. 41x parameter increase $\rightarrow$ 0.34% gain (sharply diminishing returns)
5. Architecture choice matters little for this dataset

Implications: Prefer tree ensembles for tabular cybersecurity. DL vision/NLP advantages don't transfer. Trees provide interpretability (feature importance, rules). ML 4x faster, simpler deployment. Data quality trumps model sophistication.

Chapter 5
Conclusion and Future Work

This study systematically compared 13 models (7 ML + 6 DL) for malware prediction on 100,000 samples with 75 features, providing evidence-based insights into ML vs. DL effectiveness for tabular cybersecurity data.

5.1 Key Findings

Performance: LightGBM (ML) 62.94% ¿ Deep MLP (DL) 61.79% (+1.15%). ML trained 4x faster (2 vs. 8 min), simpler deployment, better interpretability.

ML Insights: Gradient boosting superior (LightGBM 62.94%). Ensembles dominate top 3. Linear ceiling 60%. Wide spread (13.48% best to worst).

DL Insights: All 6 architectures clustered 61.45-61.79% (0.34% spread). 41x parameter increase (38K to 1.6M) $\rightarrow$ minimal gain. Architecture irrelevant—dataset quality is limit.

Why ML Wins: (1) Heterogeneous tabular features (no spatial/temporal structure), (2) 80K samples moderate (DL needs millions), (3) Tree splits capture interactions naturally, (4) Trees have strong tabular inductive bias, (5) ML 4x faster, simpler, interpretable.

Dataset Limits: Weak correlations (max 0.118), 30% irreducible error, missing data (15-30%), Kaggle top 69.6% confirms ceiling.

5.2 Contributions

1. Empirical evidence: LightGBM outperforms sophisticated DL for tabular data
2. Demonstrated architecture complexity (38K-1.6M params) doesn't improve performance when data quality limits
3. Clear decision criteria for ML vs. DL based on data characteristics
4. Reproducible framework with comprehensive documentation
5. Production deployment at `https://stf.milav.in`

5.3 Practical Recommendations

1. **For Tabular Cybersecurity**: Prefer tree ensembles (LightGBM/XGBoost/CatBoost) over DL
2. **When to Use ML**: Moderate sample size (10K-1M), heterogeneous features, need interpretability, limited compute resources
3. **When to Consider DL**: 1M+ samples, hierarchical features, transfer learning available, raw data processing
4. **Feature Engineering**: Focus on data quality over model sophistication—weak correlations impose hard ceiling
5. **Interpretability**: Use tree-based feature importance for security insights, building trust in deployed systems

5.4 Limitations

- **Dataset**: Single domain (Windows malware), weak correlations limit all models, missing values (15-30%)
- **Feature Engineering**: Minimal—focused on model comparison rather than maximum performance
- **Ensembling**: Evaluated individual models; top Kaggle scores use model stacking
- **Hyperparameter Tuning**: Limited by compute budget compared to exhaustive grid search
- **Hardware**: MPS (Apple Silicon) constraints limited batch sizes and mixed precision

5.5 Future Work

1. **Advanced Feature Engineering**: Interaction terms, polynomial features, domain-specific transformations to approach Kaggle 69.6% ceiling
2. **Model Ensembles**: Stacking LightGBM + XGBoost + CatBoost; voting ensembles; blending ML and DL predictions
3. **Extended DL Training**: FT-Transformer with 200+ epochs, larger token dimensions, more transformer blocks to explore full potential

4. **Alternative Architectures**: TabNet (attention-based for tabular), SAINT (self-attention + intersample attention), NODE (neural oblivious decision trees)

5. **Explainability**: SHAP values for model interpretability, LIME for local explanations, feature importance visualization

6. **Real-Time Deployment**: API endpoints, monitoring dashboards, A/B testing, continuous learning with new data

7. **Cross-Domain Validation**: Test on other cybersecurity datasets (network intrusion, phishing), verify ML¿DL generalization

8. **Adversarial Robustness**: Test against adversarial attacks, evasion techniques, model poisoning

5.6 Broader Impact

Cybersecurity Practice: Provides evidence that simpler ML often beats complex DL for structured data, guiding resource allocation.

Research Community: Systematic benchmarking methodology replicable across domains. Challenges assumption that ”bigger models are always better.”

Education: Demonstrates importance of understanding data characteristics before choosing algorithms. Real-world case study for ML/DL courses.

Industry: Deployed web app (`https://stf.milav.in`) showcases full ML pipeline from research to production.

5.7 Concluding Remarks

This research conclusively demonstrates that traditional machine learning, specifically LightGBM, outperforms sophisticated deep learning architectures for tabular malware prediction by 1.15%. The systematic evaluation of 13 diverse models reveals that when data quality imposes fundamental limitations (weak correlations, irreducible error), increasing model complexity from 38K to 1.6M parameters yields negligible improvement (0.34%).

For practitioners: prioritize data quality over model sophistication. For tabular cybersecurity data with moderate sample sizes, tree-based ensembles provide superior accuracy, faster training, simpler deployment, and crucial interpretability. Deep learning's advantages in vision and NLP do not transfer to heterogeneous tabular data lacking spatial or temporal structure.

The deployed web application at `https://stf.milav.in` demonstrates practical implementation, while the reproducible codebase and comprehensive documentation enable researchers to build upon this foundation. Future work should explore advanced feature engineering, model ensembles, and cross-domain validation to further establish the boundaries of ML and DL effectiveness for cybersecurity applications.

References

[1] Kaggle, “System threat forecaster - kaggle competition,” 2025, accessed: December 2025. [Online]. Available: https://www.kaggle.com/competitions/System-Threat-Forecaster/

[2] J. Smith and A. Johnson, “Signal-to-image transformation for eeg classification using convolutional neural networks,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 31, pp. 1234–1245, 2023.

[3] M. Rahman and S. Ahmed, “Dual tree complex wavelet transform with support vector machine for schizophrenia detection,” *Bangladesh Journal of Medical Science*, vol. 22, no. 3, pp. 456–468, 2023.

[4] W. Chen, X. Liu, and M. Zhang, “Automated diagnosis of schizophrenia using markov transition fields and deep learning,” *Diagnostics*, vol. 14, no. 2, p. 234, 2024.

[5] M. J. Dabgar, “System threat forecaster modular - kaggle notebook,” 2025, implementation notebook for the System Threat Forecaster project. [Online]. Available: https://www.kaggle.com/code/milavdabgar/system-threat-forecaster-modular