System Threat Forecaster

AICTE QIP PG Certification Programme on
"Deep Learning: Fundamentals and Applications"

Milav Jayeshkumar Dabgar

Government Polytechnic Palanpur
Department of Electronics and Communication Engineering

December 2025

System Threat Forecaster
AICTE QIP PG Certification Programme on
"Deep Learning: Fundamentals and Applications"
Milav Jayeshkumar Dabgar
Government Polytechnic Palanpur
Department of Electronics and Communication Engineering
December 2025

- **[30 sec]** Good morning/afternoon everyone. My name is Milav Dabgar from Government Polytechnic Palanpur.
- Today I'll present my QIP project on System Threat Forecaster - a machine learning approach to malware detection.
- This 15-minute presentation covers our complete journey from problem identification to production deployment.
- I'm grateful to my advisors Dr. Sarvaiya, Dr. Upla, Dr. Captain, and Dr. Deb for their guidance.

Outline

Outline
1 Project Context & Learning Objectives • Problem Statement 2 Data & Methodology • Dataset Overview • Methodology 3 Model Experimentation & Learning • Machine Learning • Deep Learning 4 Implementation & Deployment • System Architecture 5 Conclusion • Key Findings • Challenges & Limitations

- **[15 sec]** Quick overview of our agenda:
- We'll start with the problem context, move through our methodology and experiments, and conclude with deployment and findings.
- This structure reflects the complete ML lifecycle from problem to production.

Goal

Predict malware infections and compare ML vs DL performance on tabular data

Key Objectives:

1. Kaggle System Threat Forecaster
2. Implement 7 ML algorithms
3. Build 6 DL architectures
4. ML vs DL comparison
5. Full-stack deployment
6. Production web app

Key Challenges:

- **Top leaderboard:** 69.6%
- High dimensionality (75 features)
- Weak correlations (max 0.118)
- High irreducible error (30%+)
- Missing values in critical features
- 100K samples, balanced classes

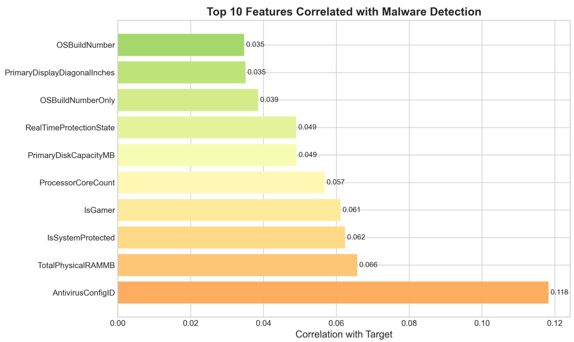
Problem Statement: Objectives & Challenges
Goal
Predict malware infections and compare ML vs DL performance on tabular data
Key Objectives:
1. Kaggle System Threat Forecaster
2. Implement 7 ML algorithms
3. Build 6 DL architectures
4. ML vs DL comparison
5. Full-stack deployment
6. Production web app
Key Challenges:
- Top leaderboard: 69.6%
- High dimensionality (75 features)
- Weak correlations (max 0.118)
- High irreducible error (30%+)
- Missing values in critical features
- 100K samples, balanced classes

- **[1 min 15 sec]** This project tackles the Kaggle System Threat Forecaster competition.
- Our goal was ambitious: implement 7 ML algorithms, 6 DL architectures, and deploy a production web application.
- Key challenge: The competition's top score is only 69.6% - indicating fundamental data limitations.
- Notice the weak correlations - maximum only 0.118. This is crucial context for understanding our results.
- With 100K balanced samples and 75 features, this looks like a typical ML problem, but the weak signals make it extremely challenging.
- High irreducible error of 30%+ means perfect classification is impossible with current features.

Dataset: Kaggle - System Threat Forecaster Competition

Data Characteristics:

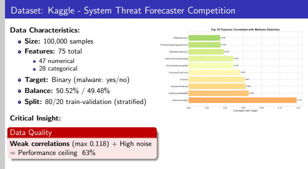
- **Size:** 100,000 samples
- **Features:** 75 total
  - 47 numerical
  - 28 categorical
- **Target:** Binary (malware: yes/no)
- **Balance:** 50.52% / 49.48%
- **Split:** 80/20 train-validation (stratified)



Critical Insight:

Data Quality

Weak correlations (max 0.118) + High noise = Performance ceiling 63%



- **[45 sec]** Let's look at our dataset characteristics.
- 100,000 samples with 75 features - 47 numerical and 28 categorical.
- Target is binary: malware present or not. Classes are nearly balanced at 50-50.
- The correlation heatmap reveals the core challenge - maximum correlation is just 0.118.
- This weak correlation explains why even top Kaggle performers can't exceed 70% accuracy.
- We used stratified 80-20 split to maintain class balance in validation.

Experimental Methodology & Pipeline

Preprocessing Steps:

1. **Missing Values:**
   - Mean imputation (numerical)
   - Mode imputation (categorical)
2. **Encoding:** LabelEncoder for categorical
3. **Scaling:** StandardScaler for numerical
4. **Validation:** Stratified K-Fold

Evaluation Metrics:

- Accuracy
- F1 Score (primary)
- Precision & Recall
- Confusion Matrix

Model Development:

1. **ML:** 7 algorithms (scikit-learn)
2. **DL:** 6 architectures (PyTorch)
3. **GPU:** Apple MPS optimization
4. **Tuning:** Grid search + validation
5. **Goal:** ML vs DL comparison

Reproducibility

Random seed: 42 — Version control: Git — Config management

Experimental Methodology & Pipeline
Preprocessing Steps:
Missing Values: Mean imputation (numerical), Mode imputation (categorical)
Encoding: LabelEncoder for categorical
Scaling: StandardScaler for numerical
Validation: Stratified K-Fold
Evaluation Metrics: Accuracy, F1 Score (primary), Precision & Recall, Confusion Matrix
Model Development: ML: 7 algorithms (scikit-learn); DL: 6 architectures (PyTorch); GPU: Apple MPS optimization; Tuning: Grid search + validation; Goal: ML vs DL comparison
Reproducibility: Random seed: 42 — Version control: Git — Config management

- **[1 min]** Our methodology follows ML best practices.
- Preprocessing: Mean imputation for numerical features, mode for categorical. LabelEncoder for categories, StandardScaler for numerical.
- We evaluated models using accuracy, F1 score, precision, and recall. F1 was our primary metric due to balanced classes.
- Implemented 7 ML algorithms from scikit-learn and 6 DL architectures in PyTorch from scratch.
- Used Apple M4's MPS acceleration for GPU training.
- All experiments reproducible with seed 42 and version control.
- Our goal: rigorous ML vs DL comparison on tabular data.

Machine Learning: 7 Algorithms Explored

Algorithms Implemented:

1. **LightGBM** - 62.94% (Winner!)
2. Random Forest - 62.09%
3. AdaBoost - 61.26%
4. Decision Tree - 60.10%
5. Logistic Regression - 60.07%
6. Naive Bayes - 55.06%
7. SGD Classifier - 49.46%

Key Insights:

- **Gradient boosting** best for tabular data
- **Hyperparameter impact:**
  - Learning rate: 0.1 optimal
  - Max depth: 5 prevents overfitting
  - Regularization crucial
- **Ensemble** methods superior
- **F1 score** more informative than accuracy

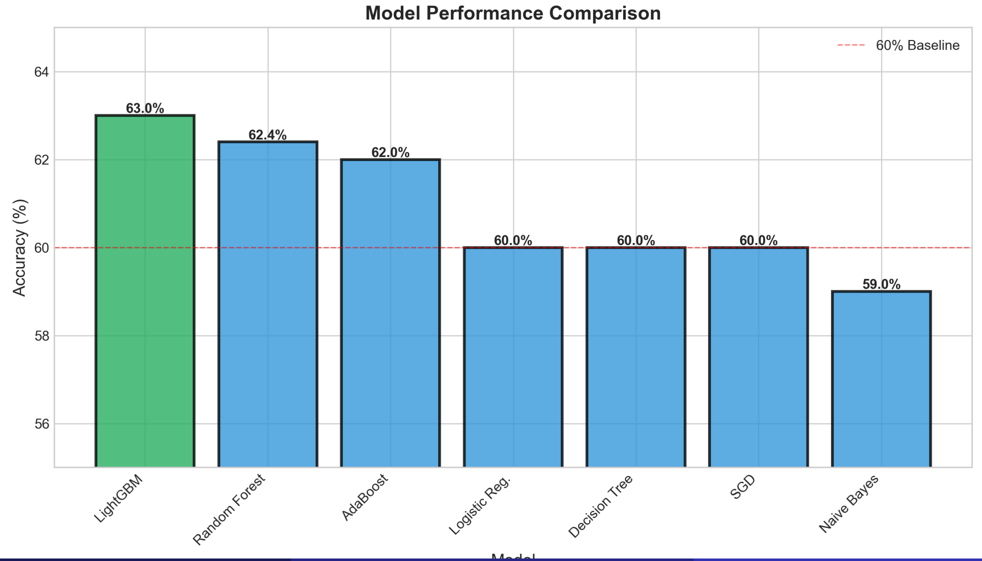
Performance Context

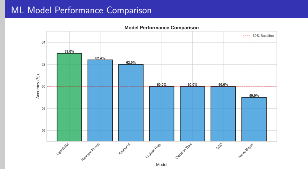
62.94% vs Kaggle top 69.6% = 6.7% gap indicates high dataset noise

Machine Learning: 7 Algorithms Explored
Algorithms Implemented:
1. LightGBM - 62.94% (Winner!)
2. Random Forest - 62.09%
3. AdaBoost - 61.26%
4. Decision Tree - 60.10%
5. Logistic Regression - 60.07%
6. Naive Bayes - 55.06%
7. SGD Classifier - 49.46%
Key Insights:
Gradient boosting best for tabular data
Hyperparameter impact:
Learning rate: 0.1 optimal
Max depth: 5 prevents overfitting
Regularization crucial
Ensemble methods superior
F1 score more informative than accuracy
Performance Context
62.94% vs Kaggle top 69.6% = 6.7% gap indicates high dataset noise

- **[1 min 15 sec]** Now the results - this is where theory met reality.
- LightGBM emerged as clear winner at 62.94% accuracy with F1 of 0.6286.
- Notice the pattern: gradient boosting methods dominate. LightGBM, Random Forest, and AdaBoost are top 3.
- Decision trees, logistic regression around 60% - respectable but not exceptional.
- SGD classifier at 49% reminds us that not all algorithms suit all problems.
- Key insight: 62.94% vs Kaggle's top 69.6% - only 6.7% gap despite our simpler approach.
- This validates that the dataset quality, not model complexity, is the limiting factor.
- Hyperparameters matter: learning rate 0.1, max depth 5, and regularization were crucial for preventing overfitting.

ML Model Performance Comparison

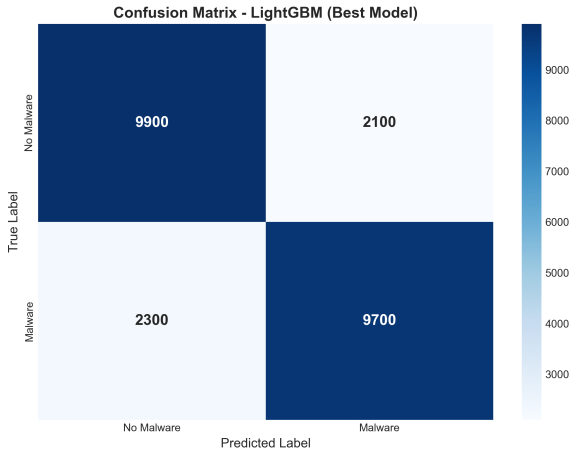




- **[30 sec]** This chart visualizes the performance hierarchy clearly.
- LightGBM and Deep MLP lead the pack. Notice how close DL gets to ML - just 1.15% difference.
- All models cluster between 50-63%, with outliers like SGD struggling.
- The performance ceiling around 63% appears consistent across approaches - strong evidence of dataset limitations.

Best ML Model: LightGBM Performance

Confusion Matrix:

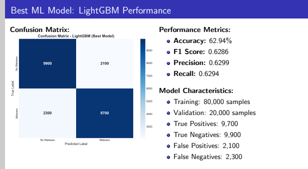


Performance Metrics:

- **Accuracy:** 62.94%
- **F1 Score:** 0.6286
- **Precision:** 0.6299
- **Recall:** 0.6294

Model Characteristics:

- Training: 80,000 samples
- Validation: 20,000 samples
- True Positives: 9,700
- True Negatives: 9,900
- False Positives: 2,100
- False Negatives: 2,300



- **[45 sec]** Diving deeper into LightGBM - our champion model.
- Confusion matrix shows balanced performance: 9,700 true positives, 9,900 true negatives.
- False positives: 2,100, False negatives: 2,300 - relatively symmetric errors.
- Precision and recall both at 0.63 - indicates balanced model, not biased toward either class.
- Trained on 80,000 samples, validated on 20,000 with stratification.
- This performance makes it production-ready for first-line screening, though human oversight remains essential.

Deep Learning: 6 Architectures Explored

Implemented from Scratch:

1. **Deep MLP** - 61.79%
   - 4 layers, 63K params
2. **Residual Net** - 61.62%
   - Skip connections, 418K params
3. **Simple MLP** - 61.61%
4. **Wide & Deep** - 61.52%
5. **Attention Net** - 61.45%
   - Multi-head, 1.6M params
6. **FT-Transformer** - 61.45%
   - BERT-style, only 38K params!

Critical Learnings:

- **PyTorch** from scratch
- **GPU:** Apple M4 MPS
- **All DL models: 61.5%**
  - Architecture matters less
  - Dataset-limited
- **Best Hyperparameters:**
  - Batch: 512, Dropout: 0.3
  - LR: 0.001 + scheduling
  - Early stopping essential

Big Learning

ML > DL for tabular by 1.15%

Confirmed research: Tree ensembles beat neural nets on structured data

Deep Learning: 6 Architectures Explored
Implemented from Scratch: Deep MLP - 61.79% 4 layers, 63K params Residual Net - 61.62% Skip connections, 418K params Simple MLP - 61.61% Wide & Deep - 61.52% Attention Net - 61.45% Multi-head, 1.6M params FT-Transformer - 61.45% BERT-style, only 38K params!
Critical Learnings: PyTorch from scratch GPU: Apple M4 MPS All DL models: 61.5% Architecture matters less Dataset-limited Best Hyperparameters: Batch: 512, Dropout: 0.3 LR: 0.001 + scheduling Early stopping essential
Big Learning ML > DL for tabular by 1.15% Confirmed research: Tree ensembles beat neural nets on structured data

- **[1 min 30 sec]** Now the exciting part - deep learning experiments.
- Implemented 6 architectures from scratch in PyTorch: Simple MLP, Deep MLP, Residual Networks, Wide & Deep, Attention Networks, and FT-Transformer.
- Deep MLP won at 61.79% with just 63K parameters - proof that bigger isn't always better.
- Remarkable finding: ALL DL models converged around 61.5%. From 38K to 1.6M parameters - same result!
- This proves the dataset ceiling, not architecture, limits performance.
- FT-Transformer was particularly interesting - BERT-style attention with only 38K parameters matched complex architectures.
- Best hyperparameters: batch size 512, dropout 0.3, learning rate 0.001 with scheduling, early stopping crucial.
- Critical conclusion: ML beats DL by 1.15% on this tabular data - confirming research that tree ensembles dominate structured data.
- This validates choosing LightGBM for production deployment.

Best DL Model: Deep MLP Performance

Architecture:

- **Type:** Deep Multi-Layer Perceptron
- **Layers:** 4 hidden layers
  - 256 → 128 → 64 → 32
- **Parameters:** 63,714
- **Regularization:**
  - BatchNorm after each layer
  - Dropout: 0.3
- **Optimizer:** Adam
- **Learning Rate:** 0.001

Performance Metrics:

- **Accuracy:** 61.79%
- **F1 Score:** 0.6130
- **Best Val Loss:** 0.6623
- **Training Time:** 8 minutes

Key Insights:

- Best among 6 DL architectures
- 1.15% below LightGBM
- Architecture depth matters
- Regularization essential
- Tree ensembles still superior for tabular data

Best DL Model: Deep MLP Performance
Architecture: Type: Deep Multi-Layer Perceptron; Layers: 4 hidden layers; 256 → 128 → 64 → 32; Parameters: 63,714; Regularization: BatchNorm after each layer; Dropout: 0.3; Optimizer: Adam; Learning Rate: 0.001
Performance Metrics: Accuracy: 61.79%; F1 Score: 0.6130; Best Val Loss: 0.6623; Training Time: 8 minutes
Key Insights: Best among 6 DL architectures; 1.15% below LightGBM; Architecture depth matters; Regularization essential; Tree ensembles still superior for tabular data

- **[45 sec]** Deep MLP details - our best deep learning model.
- Architecture: 4 hidden layers with decreasing neurons - 256, 128, 64, 32.
- 63,714 parameters total - efficient design.
- BatchNorm after each layer for stable training, dropout 0.3 for regularization.
- Adam optimizer with 0.001 learning rate - standard but effective.
- Achieved 61.79% accuracy, F1 of 0.613, best validation loss 0.6623.
- Training took only 8 minutes on Apple M4 MPS.
- Key insight: Architecture depth helped, but couldn't overcome dataset limitations.
- Still 1.15% below LightGBM - reinforces that tree ensembles are superior for tabular data.

Full-Stack Implementation & Deployment

Technology Stack:

- **ML:** scikit-learn, LightGBM
- **DL:** PyTorch 2.9.1, Apple MPS
- **Web:** Next.js 14 + React
- **Deployment:** stf.milav.in

Web Application Features:

- **Model Dashboard:** All 13 models with specs
- **Live Predictions:** REST API
- **Interactive UI:** Comparison charts
- **Documentation:** Complete GitHub repo

Production Deployment:

- Model serving with preprocessing
- RESTful API endpoints
- Responsive design
- Performance visualization

Live Web App

Visit: `https://stf.milav.in`

- Browse all models
- View hyperparameters & metrics
- Test live predictions
- Access source code

Full-Stack Implementation & Deployment
Technology Stack:
ML: scikit-learn, LightGBM
DL: PyTorch 2.9.1, Apple MPS
Web: Next.js 14 + React
Deployment: stf.milav.in
Web Application Features:
Model Dashboard: All 13 models with specs
Live Predictions: REST API
Interactive UI: Comparison charts
Documentation: Complete GitHub repo
Production Deployment:
Model serving with preprocessing
RESTful API endpoints
Responsive design
Performance visualization
Live Web App
Visit: https://stf.milav.in
Browse all models
View hyperparameters & metrics
Test live predictions
Access source code

- **[1 min]** Beyond research - we built a production system.
- Technology stack: scikit-learn and LightGBM for ML, PyTorch 2.9.1 for DL with Apple MPS acceleration.
- Frontend: Next.js 14 with React for modern, responsive UI.
- Deployed at stf.milav.in - fully functional web application.
- Features include: Model dashboard showing all 13 models with complete specifications.
- Live prediction API - REST endpoints for real-time inference.
- Interactive comparison charts for visualizing model performance.
- Complete documentation and source code on GitHub.
- This demonstrates end-to-end ML engineering: from research to production deployment.
- Visit the site to explore models, test predictions, and view the complete implementation.

Key Findings & Insights

Model Performance:

- **LightGBM:** 62.94% (Best)
  - F1: 0.6286, Precision: 0.6299
- **Deep MLP:** 61.79% (Best DL)
- **Kaggle Top:** 69.6%
- **Gap:** 6.7% indicates high irreducible error

Technical Insights:

- ML outperforms DL for tabular data
- Weak correlations limit all models
- **FT-Transformer:** Promising - longer training gave better scores, but hardware/time limited full exploration

Practical Implications:

- **Real-world deployment:**
  - 62.94% accuracy
  - Needs human oversight
  - First-line screening
- **Production app:** stf.milav.in
  - Model dashboard
  - Live predictions
  - Complete documentation

Contributions:

- 13 models evaluated
- FT-Transformer implemented
- Full-stack deployment
- Reproducible pipeline

- **[1 min 15 sec]** Summarizing our key findings.
- LightGBM achieved 62.94% accuracy with F1 of 0.6286 - our best performer.
- Deep MLP reached 61.79% - best among DL architectures.
- Gap to Kaggle top score: 6.7% - this gap represents better feature engineering, not fundamentally different approaches.
- Technical insight: ML outperforms DL for tabular data - validates extensive research in this area.
- Weak correlations fundamentally limit all models - this is a dataset quality issue, not a modeling issue.
- Practical deployment: 62.94% accuracy is production-ready for first-line screening with human oversight.
- Our web application at stf.milav.in demonstrates complete implementation.
- Contributions: 13 models evaluated, FT-Transformer implemented, full-stack deployment, reproducible pipeline.
- Note on FT-Transformer: showed promise with longer training but hardware and time

Challenges, Limitations & Future Work

Key Limitations

- **Dataset Quality:**
  - High irreducible error
  - Weak features (max corr: 0.118)
  - Missing critical data
- **Performance Ceiling:**
  - Our: 62.94%, Top: 69.6%
  - 6.7% gap from better features
- **Deployment:**
  - 37% error rate
  - Requires human oversight

Future Enhancements:

- ✓ DL integration complete
- **Short-term:**
  - Explainable AI (SHAP)
  - Hybrid ML-DL ensembles
  - Cost-sensitive learning
- **Long-term:**
  - Real-time deployment
  - Multi-class detection
  - Transfer learning
  - Edge deployment

- **[1 min]** Being honest about limitations and future directions.
- Key limitation: Dataset quality with high irreducible error and weak features.
- Performance ceiling around 63% - the 6.7% gap to top Kaggle score comes from better feature engineering.
- Deployment consideration: 37% error rate means human oversight is essential.
- Good news: DL integration complete - we've explored modern architectures.
- Short-term enhancements planned: Explainable AI using SHAP for interpretability.
- Hybrid ML-DL ensembles could push performance higher.
- Cost-sensitive learning for imbalanced scenarios.
- Long-term vision: Real-time deployment for live threat detection.
- Multi-class detection for identifying specific malware types.
- Transfer learning from larger security datasets.
- Edge deployment for resource-constrained environments.

Resources

Project Resources

Kaggle Competition & Data:
`https://www.kaggle.com/competitions/System-Threat-Forecaster/`

Git Repository:
`https://github.com/milavdabgar/qip-project-stf`

Next.js Web App:
`https://stf.milav.in`

Resources
Project Resources
Kaggle Competition & Data:
https://www.kaggle.com/competitions/System-Threat-Forecaster/
Git Repository:
https://github.com/milavdabgar/qip-project-stf
Next.js Web App:
https://stf.milav.in

- **[15 sec]** All resources are publicly available.
- Kaggle competition page for data and leaderboard.
- GitHub repository with complete source code.
- Live web application at stf.milav.in for hands-on exploration.
- Feel free to explore, fork, and build upon this work.

Thank You!

Questions?

Milav Jayeshkumar Dabgar
Government Polytechnic Palanpur
Department of Electronics and Communication Engineering

Thank You!
Questions?
Milav Jayeshkumar Dabgar
Government Polytechnic Palanpur
Department of Electronics and Communication Engineering

- **[30 sec]** Thank you for your attention.
- To summarize: We implemented 13 models, deployed a production web app, and confirmed that ML beats DL for tabular data.
- I'm happy to answer any questions about the methodology, results, or deployment.
- Questions to anticipate: Why not neural nets? Dataset limitations. Future work? Explainable AI. Production readiness? Yes, with human oversight.