PART-A OF THIS DOCUMENT: DESCRIPTION OF WHAT THE PAPER IS ABOUT

Title:

Comparative analysis of IDS using machine learning and deep learning

Literature survey:

1. "A Review of Intrusion Detection Systems Using Machine Learning: Attacks, Algorithms and Challenges" (Gutierrez-Garcia et al., 2023)

- **Problem/Gap:** The entire field is hindered by a **major disconnect between academic research and real-world industrial needs**. This is caused by:
    1. **Outdated Datasets:** Most research uses old datasets that don't reflect modern threats.
    2. **Lack of Robustness Testing:** Models are rarely tested against **adversarial attacks** designed to fool them.
    3. **Lack of Industry Collaboration:** Research is often done in a vacuum without access to real-world data and challenges.

2. "Overview on Intrusion Detection Systems Design Exploiting Machine Learning for Networking Cybersecurity" (Dini et al., 2023)

- **Problem/Gap:** The literature lacks **rigorous, large-scale experimental comparisons** of traditional ML models. Specifically, it points out:
    1. **Limited Scope of Comparison:** Many studies use only one dataset or test only a few algorithms.
    2. **Missing Efficiency Analysis:** Most papers ignore crucial practical metrics like **computational time**.
    3. **Lack of DL Benchmarks:** It intentionally creates a gap by **not testing deep learning models**, making a clear case for future work to compare DL against its thorough ML benchmarks.

3. "Comparative Analysis of Intrusion Detection Systems: Leveraging Classification Algorithms and Feature Selection Techniques" (Sulaiman & Abdulazeez, 2024)

- **Problem/Gap:** The "curse of dimensionality" is a critical problem where high numbers of features in IDS datasets degrade performance. The knowledge gap is the **lack of a systematic overview of how feature selection (FS) techniques are paired with classifiers**. It shows that the choice of FS method is just as important as the choice of the classifier itself.

4. "Comparative Analysis of Intrusion Detection Systems and Machine Learning-Based Model Analysis Through Decision Tree" (Azam et al., 2023)

- **Problem/Gap:** Many modern ML/DL models are overly complex "black boxes" that are slow and difficult for security analysts to trust and use. The knowledge gap is the need to find a model that provides an optimal balance of **performance, speed, and interpretability**. The paper argues that the **Decision Tree** fills this gap effectively. It also highlights the lack of discussion in other reviews about **attacker evasion techniques**.

5. "Deep transfer learning for intrusion detection in industrial control networks: A comprehensive review" (Kheddar et al., 2024)

- **Problem/Gap:** Specialized, critical domains like Industrial Control Networks (ICNs) suffer from a severe **scarcity of labeled data**, making it impossible to train standard deep learning models from scratch. The knowledge gap is the **lack of a comprehensive survey on Deep Transfer Learning (DTL)** as a specific solution to this data scarcity problem in IDS.

6. "A Survey Paper on Automated Intrusion Detection using Deep Learning Techniques" (Manivannan et al., 2024)

- **Problem/Gap:** Traditional IDS methods are failing to detect complex, modern threats. The knowledge gap is the need for a consolidated summary of **which specific deep learning architectures (CNN, LSTM, Hybrids) are proving most effective** on the latest benchmark datasets. It focuses on synthesizing the state-of-the-art specifically within the DL paradigm.

7. "A Review of Deep Learning Applications in Intrusion Detection Systems: Overcoming Challenges..." (Zhang et al., 2025)

- **Problem/Gap:** Applying DL to IDS is not a simple plug-and-play task. The knowledge gap lies in understanding how to overcome two specific, advanced challenges:
    1. **Spatiotemporal Features:** How to design models that can understand the complex time- and space-based patterns in network traffic.
    2. **Data Imbalance:** How to handle the fact that attack data is extremely rare compared to normal data, which biases models.

8. "Comparative Analysis of Intrusion Detection System Using Machine Learning and Deep Learning Algorithms" (Note & Ali, 2022)

- **Problem/Gap:** There is a need for a broad, empirical benchmark to understand the practical **trade-off between performance and implementation time**. The knowledge gap is the lack of studies that test a very wide range of *both* ML and DL algorithms across multiple datasets to see which ones offer the best practical balance (e.g., Random Forest is most accurate, but Decision Tree is much faster and still very accurate).

9. "Deep Learning vs. Machine Learning for Intrusion Detection in Computer Networks: A Comparative Study" (Ali et al., 2025)

- **Problem/Gap:** While it's often claimed that Deep Learning is superior to Machine Learning for IDS, there is a gap in **direct, head-to-head experimental comparisons on modern, imbalanced datasets**. This paper aims to fill that gap by explicitly using SMOTE to handle imbalance and testing both model types on the same data (CICIDS2017).

10. "A Comparative Analysis of Supervised and Unsupervised Models for Detecting Attacks on..." (Khoei & Kaabouch, 2023)

- **Problem/Gap:** The vast majority of IDS research focuses on supervised learning, which requires labeled data. The knowledge gap is the **lack of comparative analysis between supervised and unsupervised learning models** for IDS. The paper aims to determine how well unsupervised methods (which don't need labeled data) perform relative to their supervised counterparts.

The research gap I am trying to bridge

1. Select a latest dataset on IDS strictly from 2023 or 2024 which is publicly available such that there are no papers who have worked on this dataset in the way that I am planning to work
2. The comparisons that I am making would be Interpretability-Performance Trade-off Analysis where each model would be compared on the basis of Detection performance, Computational efficiency, Interpretability score (using SHAP/LIME). This would include 8 models/algorithms of ml and dl.
3. The aim of this is basically select a dataset which has not been used for similar study to this, so that my paper would have contribution and might server as a benchmark for this particular dataset

Exact Performance metrics that I am planning to use:

1. Standard Performance Metrics

- Accuracy
- Precision
- Recall (Sensitivity, TPR)
- F1-Score

2. Error & Class Imbalance-Sensitive Metrics

- False Positive Rate (FPR)
- False Negative Rate (FNR)
- Specificity (True Negative Rate, TNR)
- Balanced Accuracy
- Misclassification Rate
- Per-class metrics (e.g., class-wise Precision/Recall/F1)

3. Imbalanced Data-Oriented Metrics

- AUC-ROC (Area under Receiver Operating Characteristic curve)
- AUC-PR (Area under Precision-Recall curve)
- G-Mean (Geometric Mean of TPR & TNR)
- Matthews Correlation Coefficient (MCC)

4. Efficiency / Resource Metrics

- Training time
- Inference time
- Memory usage
- Energy consumption (sustainable AI focus)

5. Interpretability & Explainability Metrics

- SHAP-based feature importance scores
- LIME explanation consistency
- Model complexity score (e.g., decision tree depth, NN parameters)
- Explanation fidelity score (how well explanation reflects true model behavior)

Dataset:

- TII-SSRC-23 (2023, publicly available). https://www.kaggle.com/datasets/daniaherzalla/tii-ssrc-23
- Rich set of benign & malicious network traffic flows with realistic attack diversity.
- Imbalanced class distribution — requires class balancing and careful per-class evaluation.
- Chosen because it is recent, under-explored in interpretability research, and fits IDS scope.

Algorithms:

A spectrum from interpretable → high-performance → cutting-edge:

1. **Logistic Regression** → linear, interpretable baseline.
2. **Decision Tree (CART)** → rule-based, transparent decisions.
3. **Random Forest** → ensemble of trees, robust + feature importance.
4. **XGBoost** → state-of-the-art gradient boosting, high accuracy.
5. **SVM (linear & RBF)** → strong traditional ML baseline.
6. **Feed-forward Neural Network (MLP)** → deep non-linear baseline for tabular data.
7. **Transformer Encoder** → models temporal/sequential traffic patterns.
8. **Graph Neural Network (GNN)** → models host–flow relationships, detects relational attacks.

Model-wise explanation methods

1. **Tree-based models (Decision Tree, Random Forest, XGBoost)**
   - **TreeSHAP**: a variant of SHAP optimized for tree ensembles.
   - Why? It computes exact SHAP values for trees (fast + mathematically correct), unlike approximations needed for deep models.
   - Outcome: tells you exactly how much each feature contributes to a prediction.
2. **Linear models (Logistic Regression)**
   - **Coefficient analysis**: just look at the learned weights — since linear models are directly interpretable.
   - **SHAP (LinearExplainer)**: gives per-sample attribution in probability space.
   - Why? This way you can compare it on the same interpretability scale as other models.
3. **SVM (Support Vector Machine)**
   - Not naturally interpretable (esp. RBF kernel).
   - Use **LIME** or **KernelSHAP** (model-agnostic methods).
   - Why? These approximate the model locally (by perturbing input and fitting a simple surrogate) to show which features drove the decision.
4. **MLP (Feed-forward Neural Network) & Transformer Encoder**
   - **DeepSHAP** (a variant of SHAP adapted for deep nets).
   - **Integrated Gradients (IG)**: a gradient-based attribution method.
   - **Attention visualization (for Transformer)**: look at attention weights to see which parts of the input the model focused on.
   - Why? These methods handle the non-linear, layered structure of neural nets.
5. **GNN (Graph Neural Network)**
   - **GNNExplainer**: identifies the most important nodes, edges, and features for a given prediction.
   - **GraphSHAP**: applies SHAP principles to graph structures.
   - Why? Standard SHAP/LIME don't work well for graph-structured data; you need specialized XAI designed for graphs.

Major questions and their answers related to my topic:

1. What is your dataset choice (it should be a latest dataset which does not have similar work to mine done on it since this is one gap I am trying to bridge): **TII-SSRC-23**
2. 2.1. what are we comparing and what is unique in it as almost all papers compare same things: **Interpretability-Performance Trade-off Analysis**
3. what algorithms to choose: **answered**
4. what evaluation metrics will be chosen: **answered**
5. what's unique in our paper that will distinguish it: **baseline paper for a latest dataset, comparison of performance and interpretability's**

PART A ENDS HERE

PART B: TEAM HANDBOOK ABOUT THE PAPER (Group Context Document for IDS Research) Project

1. Topic

Our research focuses on **Intrusion Detection Systems (IDS) using Machine Learning (ML) and Deep Learning (DL)**.

- IDS are systems that monitor network traffic to detect **malicious activity or attacks**.
- The field combines **cybersecurity** and **data science**, using AI/ML models to classify network events as normal or malicious.
- We are particularly focusing on **modern datasets** (TII-SSRC-23, 2023), which contain **realistic attack patterns** and **imbalanced traffic classes**, reflecting current cybersecurity challenges.

2. Desired Goal

The **main objective** is to perform a **systematic comparison of ML and DL models** for IDS, emphasizing both:

1. **Detection performance** – How accurately the models detect attacks.
2. **Interpretability** – How understandable the model decisions are to a human analyst.
3. **Efficiency** – How resource-intensive the models are (time, memory, energy).

We want to **bridge a research gap** by creating a **baseline benchmark on TII-SSRC-23** that combines all three aspects (performance, interpretability, efficiency) in one study.

3. What We Are Going to Do

1. Select **8 ML/DL models** covering interpretable, high-performing, and cutting-edge algorithms:
   - Logistic Regression, Decision Tree, Random Forest, XGBoost, SVM, Feed-forward Neural Network, Transformer Encoder, Graph Neural Network (GNN).
2. Train and evaluate these models on **TII-SSRC-23**, a 2023 publicly available IDS dataset.

3. Measure **performance, efficiency, and interpretability** using a comprehensive set of metrics.
4. Analyze **trade-offs** between performance, interpretability, and computational cost.

4. How We Are Doing It

Step-by-step approach:

1. **Dataset preparation**:
    - Download TII-SSRC-23. https://www.kaggle.com/datasets/daniaherzalla/tii-ssrc-23
    - Clean, preprocess, and handle imbalanced classes (SMOTE, class-weighting, etc.).
    - Split into stratified train/validation/test sets.
2. **Model training**:
    - Train all 8 selected models with hyperparameter tuning.
    - Record training time, memory usage, and compute resource usage.
3. **Performance evaluation**:
    - Compute standard metrics (Accuracy, Precision, Recall, F1).
    - Compute class imbalance-aware metrics (FPR, FNR, Specificity, Balanced Accuracy, MCC, G-Mean).
    - Compute ROC-AUC, PR-AUC, and per-class metrics.
4. **Interpretability analysis**:
    - Tree models → TreeSHAP.
    - Linear models → Coefficients + Linear SHAP.
    - SVM → LIME / KernelSHAP.
    - Neural Networks & Transformer → DeepSHAP / Integrated Gradients, attention maps.
    - GNN → GNNExplainer / GraphSHAP.
    - Compute explanation fidelity, stability, feature importance, and model complexity scores.
5. **Efficiency analysis**:
    - Record training and inference times, memory consumption, and energy usage.
6. **Trade-off analysis**:
    - Compare all models across performance, interpretability, and efficiency.
    - Identify models that are **best overall**, and models that offer **good trade-offs** between accuracy and explainability.

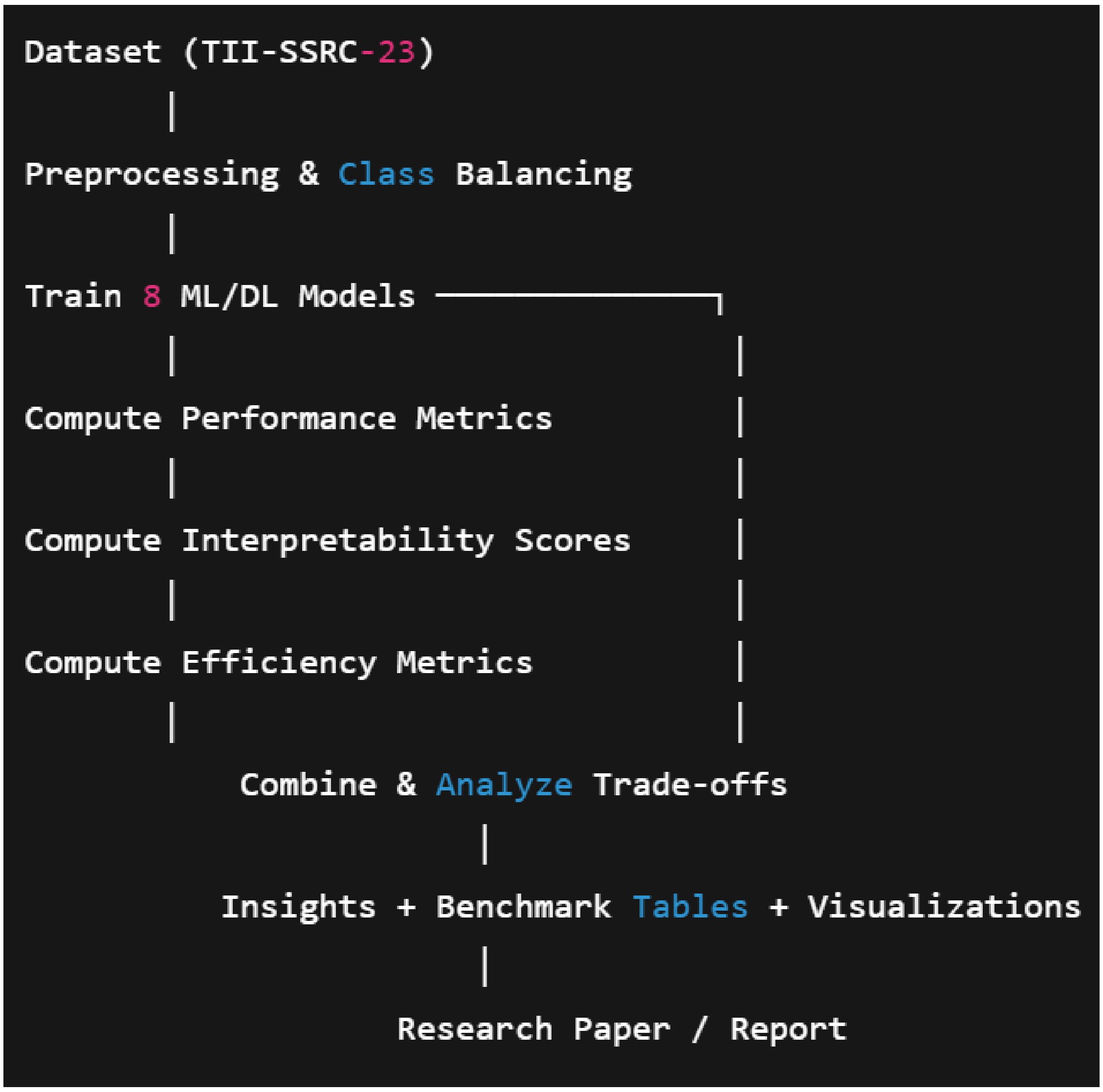
5. Expected Output

1. **A comprehensive benchmark** of ML/DL models on TII-SSRC-23.
2. **Tables and visualizations** showing:
    - Model performance per metric.
    - Interpretability scores (SHAP/LIME analyses).

- Efficiency metrics (time, memory, energy).
- Trade-off comparisons across models.

3. **Insights** about which models are suitable in real-world IDS deployments considering accuracy, interpretability, and computational cost.
4. **A potential baseline reference** for future researchers using TII-SSRC-23.

6. Short Flow of Entire Process



```
Dataset (TII-SSRC-23)
       |
Preprocessing & Class Balancing
       |
Train 8 ML/DL Models ──────────────┐
       |                             |
Compute Performance Metrics          |
       |                             |
Compute Interpretability Scores      |
       |                             |
Compute Efficiency Metrics           |
       |                             |
           Combine & Analyze Trade-offs
                       |
          Insights + Benchmark Tables + Visualizations
                       |
                    Research Paper / Report
```

7. Key Points Every Member Should Know

- TII-SSRC-23 is **new, realistic, and under-explored** for interpretability studies.
- We are **not just looking for accuracy**; explainability and resource usage are equally important.
- Interpretability tools **must match the model type** (TreeSHAP for trees, IG for NNs, GNNExplainer for GNN).
- All experiments must be **reproducible** (documented splits, seeds, hardware/software).
- The **final contribution** is a clear, unified benchmark for performance–interpretability–efficiency trade-offs.

PART B ENDS HERE

PART C : Comprehensive Model Development Guide for IDS Research

Step 1: Setup Environment

- Open **Google Colab** and create a new notebook.
- Install all necessary libraries for data processing, machine learning, deep learning, and interpretability.
- Import libraries and explain why each is required.
- Set **random seeds** to ensure reproducibility.
- Document software and library versions so results are consistent across all team members.

Step 2: Load Dataset

- Load the chosen dataset (**TII-SSRC-23**) into the workspace. https://www.kaggle.com/datasets/daniaherzalla/tii-ssrc-23
- Understand dataset content:
  - Number of samples, features, and labels.
  - Types of features: numerical, categorical, or binary.
  - Class distribution: IDS datasets are often imbalanced.
- Document the source and relevance of the dataset to IDS research.

Step 3: Exploratory Data Analysis (EDA)

- Inspect dataset structure and quality:
  - Check for missing values, anomalies, or inconsistencies.
  - Examine descriptive statistics (mean, median, min, max) of numerical features.
- Check **class balance** to plan handling of imbalance later.
- Visualize features and relationships:
  - Histograms, boxplots, or violin plots for numerical features.
  - Count plots for categorical features.
  - Correlation heatmaps to detect redundant or highly correlated features.
- Document all insights from EDA in markdown for team understanding.

Step 4: Preprocessing

- **Handle missing values**: impute or remove depending on context.
- **Encode categorical features** for machine learning models.
- **Scale or normalize features** if required (especially for SVM, neural networks).
- **Address class imbalance**:
    - Oversampling (e.g., SMOTE), undersampling, or class weighting.
    - Document why the chosen method is appropriate.
- Split dataset into **training, validation, and test sets** with stratification to preserve class distribution.
- Document every preprocessing choice for reproducibility.

Step 5: Hyperparameter Tuning

- Identify **key hyperparameters** for each model:
    - Logistic Regression: regularization type and strength.
    - Decision Tree: max depth, min samples per leaf, split criterion.
    - Random Forest: number of trees, max features, max depth.
    - XGBoost: learning rate, number of trees, max depth, subsample ratio.
    - SVM: kernel type, C value, gamma (for RBF).
    - MLP: layers, neurons per layer, learning rate, batch size, activation.
    - Transformer Encoder: attention heads, hidden size, learning rate, dropout.
    - GNN: layers, hidden units, learning rate, aggregation method.
- Decide **search strategy**:
    - Grid search, random search, or advanced methods like Bayesian optimization.
- Evaluate each hyperparameter combination using cross-validation or a validation set.
- Document **why certain hyperparameters were selected** and how they affect performance or efficiency.

Step 6: Model Training

- Train each model using **best-found hyperparameters**.
- Track:
    - Training time and memory usage.
    - Loss and accuracy curves (for neural networks).
    - Early stopping criteria and optimizer details.
- Document training process so all team members understand each decision.
- Save the trained model after completion.

Step 7: Performance Evaluation

- Evaluate models using multiple metrics:
    1. **Standard metrics:** Accuracy, Precision, Recall, F1-score.
    2. **Error and imbalance-sensitive metrics:** FPR, FNR, Specificity, Balanced Accuracy, Misclassification Rate, per-class metrics.
    3. **Imbalanced data-oriented metrics:** ROC-AUC, PR-AUC, G-Mean, MCC.
    4. **Efficiency metrics:** Training time, inference time, memory usage, energy consumption.
    5. **Interpretability metrics:** SHAP or LIME feature importance, explanation fidelity, explanation stability, model complexity.
- Document what each metric tells you about the model's strengths and weaknesses.

Step 8: Interpretability Analysis

- Use model-specific interpretability methods:
    - **Tree models (Decision Tree, Random Forest, XGBoost):** TreeSHAP.
    - **Linear models (Logistic Regression):** Coefficient analysis + Linear SHAP.
    - **SVM:** LIME or KernelSHAP.
    - **MLP & Transformer:** DeepSHAP, Integrated Gradients, and attention visualization for Transformer.
    - **GNN:** GNNExplainer or GraphSHAP.
- Record explanations for feature contributions and check **consistency** across different runs.
- Analyze which features are most important for detecting attacks.

Step 9: Efficiency Analysis

- Track:
    - Training and inference time for each model.
    - Memory usage.
    - Energy consumption (especially for deep learning models).
- Compare efficiency alongside performance to identify **trade-offs**.

Step 10: Model Saving

- Save all trained models in formats that allow **later loading and inference**.
- Include metadata: preprocessing steps, hyperparameters, and version of libraries used.
- Ensure all team members can load models into other notebooks consistently.

Step 11: Documentation and Review

- Use detailed **comments and markdown explanations** in the notebook to describe:
    - What was done.
    - Why each step was taken.
    - Observations and insights.
- Visualize key results:
    - Metric tables.
    - SHAP/LIME feature importance plots.
    - Efficiency comparisons.
- Have all team members run the notebook independently to verify reproducibility.
- Summarize the **best models**, highlighting trade-offs between performance, interpretability, and efficiency.

PART C ENDS HERE