A Comprehensive Guide to the Development and Application of Explainable Feature-Weighted K1K2NN (XFW-K1K2NN) for Network Intrusion Detection

Section I: Algorithmic Foundations of XFW-K1K2NN

The escalating complexity and volume of network traffic, coupled with the ever-evolving landscape of cyber threats, necessitate the development of Network Intrusion Detection Systems (NIDS) that are not only accurate but also transparent and trustworthy. Traditional machine learning models often operate as "black boxes," providing predictions without clear rationale, which severely limits their utility in a security operations context where understanding the *why* behind an alert is as critical as the alert itself.[1] The Explainable Feature-Weighted K1K2NN (XFW-K1K2NN) algorithm is proposed as a novel framework designed to address these dual challenges of performance and interpretability. This section establishes the theoretical bedrock of the XFW-K1K2NN model by deconstructing its constituent components: the foundational k-Nearest Neighbors (kNN) algorithm, the innovative two-stage K1K2NN structure, the performance-enhancing mechanism of feature weighting, and the critical integration of an explainability layer. By tracing the origins of each component and justifying their synthesis, this section lays the groundwork for the comprehensive methodological framework presented in subsequent chapters.

1.1 The k-Nearest Neighbors (kNN) Paradigm: A Primer on Lazy Learning

The k-Nearest Neighbors (kNN) algorithm represents one of the most intuitive and

fundamental methods in supervised machine learning.[3] As a non-parametric technique, it makes no underlying assumptions about the distribution of the data, a property that makes it robust and broadly applicable, particularly in domains like network security where traffic patterns may not conform to standard statistical distributions.[5]

Core Principles

The central tenet of kNN is rooted in the principle of proximity: a new, unclassified data point is assigned the class label that is most common among its 'k' closest neighbors in the n-dimensional feature space.[7] This process is analogous to a simple voting system, where each of the 'k' neighbors casts a vote for its own class, and the class with the plurality of votes determines the final classification of the new instance.[3] If the hyperparameter 'k' is set to 1, the algorithm simplifies to the nearest neighbor rule, where the new instance is assigned the class of its single closest neighbor.[3] While primarily used for classification, the same principle can be extended to regression tasks, where the predicted value for a new instance is the average of the values of its 'k' nearest neighbors.[3]

Instance-Based vs. Model-Based Learning

A defining characteristic of kNN is its classification as a "lazy learning" or "instance-based" algorithm.[5] This stands in stark contrast to "eager learning" algorithms, such as Decision Trees, Support Vector Machines, or Neural Networks. Eager learners undergo a distinct training phase where they build a generalized, abstract model from the training data. This model, once built, is used to make predictions on new, unseen data, and the original training data can often be discarded.[10]

Lazy learners, conversely, defer all significant computation until the prediction (or inference) phase.[7] The "training" phase for kNN is trivial; it simply consists of storing the entire labeled training dataset in memory.[7] When a prediction is required for a new instance, the algorithm then performs the computationally intensive task of calculating the distance from this new point to every single point in the stored training set to identify its nearest neighbors.[3] This architectural choice has profound implications. While it makes the algorithm conceptually simple and allows it to adapt quickly to new data, it also introduces significant drawbacks, particularly for applications like NIDS. The memory requirement to store the entire dataset can be substantial, and the computational cost of making a single prediction scales with the size of the training set, making it inefficient for large datasets and potentially too slow for real-time

intrusion detection scenarios.[3]

Distance Metrics

The notion of "nearness" or "similarity" is the cornerstone of the kNN algorithm and is quantified using a distance metric. The choice of metric is a critical design decision that depends on the nature of the data.

- **Euclidean Distance (L2 Norm):** This is the most prevalent distance metric used in kNN, especially for continuous, real-valued features.[6] It represents the straight-line distance between two points in Euclidean space and is a direct application of the Pythagorean theorem extended to multiple dimensions.[13] For two points,
  p and q, in an n-dimensional space, the Euclidean distance is calculated as:

  $$d(p,q)=i=1\Sigma n(pi-qi)2$$

  This metric is intuitive and effective but can be sensitive to differences in feature scales and the curse of dimensionality.13
- **Manhattan Distance (L1 Norm):** Also known as city block distance, this metric calculates the distance as the sum of the absolute differences of their coordinates.[12] It is as if one were navigating a grid, restricted to moving along orthogonal paths. The formula is:
  $$d(p,q)=i=1\Sigma n|pi-qi|$$

  Manhattan distance can be preferable to Euclidean distance in high-dimensional spaces or when features are not of a similar type, as it is less sensitive to outliers because it does not square the differences.14
- **Hamming Distance:** This metric is specifically designed for categorical or binary data. It measures the number of positions at which the corresponding symbols are different.[12]
  For example, the Hamming distance between the binary vectors
  10110 and 11100 is 2. This makes it suitable for comparing network traffic features that are inherently categorical, such as protocol types or connection flags.

1.2 The K1K2NN Framework: A Two-Stage Neighboring Approach

The K1K2NN algorithm is an advanced classifier that extends the traditional kNN paradigm by incorporating a second stage of analysis that operates in the label space rather than the

instance space. Originally developed for multi-label classification problems in computational biology, its core logic offers a powerful new way to handle complex classification tasks.[18]

Conceptual Origin

The K1K2NN model was first proposed to predict the side effects of COVID-19 drugs based on their chemical properties.[18] This is a multi-label problem because a single drug can have multiple side effects. The algorithm was built on two key propositions: (1) chemically similar drugs are likely to have similar side effects, and (2) side effects that frequently co-occur can be used to inform predictions for chemically similar drugs.[18] This two-pronged approach—first finding similar instances, then finding similar labels—is what distinguishes K1K2NN from standard classification methods.

The K1 Step (Instance-Space Search)

The first stage of the K1K2NN algorithm is a direct and conventional application of kNN. For a given test sample (in the original context, a drug; in our context, a network flow), the model calculates its similarity to all samples in the training set using a suitable distance metric.[18] It then identifies the

K1 nearest neighbors—the K1 training samples that are most similar to the test sample. An initial set of labels is then assigned to the test sample based on a majority voting mechanism among these K1 neighbors. For a multi-class problem, this would typically result in the single class that appears most frequently among the neighbors being assigned as the initial prediction.[18] This step directly addresses the first proposition: that similar instances are likely to share similar labels.

The K2 Step (Label-Space Search)

The second stage is the novel contribution of the K1K2NN framework. After the K1 step has produced an initial set of predicted labels, the K2 step refines and expands this prediction by analyzing relationships between the labels themselves. The algorithm first computes a label-label similarity matrix based on their co-occurrence across the entire training dataset.[18] A common metric for this is the Jaccard similarity, which measures the similarity between

finite sample sets.[18]

For each of the initial labels identified in the K1 stage, the model then consults this similarity matrix to find the K2 most similar or most frequently co-occurring labels. These newly identified, highly correlated labels are then added to the final prediction set for the test sample.[18] This step addresses the second proposition: that co-occurring labels can be assigned to similar instances. In the original multi-label context, this allows the model to predict side effects that may not have been present in the immediate

K1 neighbors but are known to be strongly associated with the side effects that were. Adapting this powerful concept from its native multi-label domain to the multi-class domain of network intrusion detection is a primary challenge that will be addressed in Section II of this report.

1.3 Feature Weighting: Enhancing Distance with Importance

A significant and well-documented weakness of the standard kNN algorithm is its naive treatment of features. By default, every feature in the dataset contributes equally to the distance calculation, which implicitly assumes that all features are equally important for the classification task.[15] This assumption is rarely valid, especially in high-dimensional domains like network security. NIDS datasets often contain dozens or even hundreds of features, many of which may be redundant, irrelevant, or noisy.[20] When these non-informative features are included in the distance calculation, they can obscure the signal from the truly important features, degrading the model's ability to identify meaningful neighbors and ultimately reducing its classification performance.[15]

Weighted kNN and Weighted Euclidean Distance

To address this limitation, the concept of feature weighting can be integrated into the kNN framework. This can take two forms. The first, often called **Weighted kNN**, assigns a weight to the *vote* of each neighbor, typically as an inverse function of its distance to the test point. This gives closer neighbors more influence on the final classification than more distant ones.[3]

The second, more powerful approach, and the one central to the XFW-K1K2NN model, is the use of a **Weighted Distance Metric**. Instead of weighting the neighbors' votes, this method weights the contribution of each *feature* within the distance calculation itself. The most common implementation of this is the Weighted Euclidean Distance.[15] The formula modifies

the standard Euclidean distance by introducing a weight vector

w, where each element wi corresponds to the importance of the i-th feature:

$$d_w(p,q) = \sum_{i=1}^{n} w_i (p_i - q_i)^2$$

By assigning higher weights (wi) to more discriminative features and lower weights to less relevant ones, the distance metric becomes more sensitive to the dimensions that truly matter for classification. This allows the algorithm to effectively "stretch" the feature space along important axes and "compress" it along unimportant ones, leading to a more meaningful and robust definition of neighborhood. This technique is a crucial step in mitigating the "curse of dimensionality," a phenomenon where distance metrics become less meaningful in high-dimensional spaces, and is essential for achieving high performance on complex NIDS datasets.15 The challenge, then, becomes how to derive a meaningful set of weights, a problem addressed by the "FW" component of our proposed model.

1.4 The Imperative of Explainability in Network Security

In high-stakes domains such as finance, medicine, and cybersecurity, the predictive accuracy of a machine learning model is necessary but not sufficient for its adoption and effective use.[2] A security analyst who receives an alert from a NIDS must be able to trust it, understand its context, and use the information to formulate a response. A prediction from a "black-box" model, which provides an answer with no justification, is often unactionable. It leaves the analyst with critical unanswered questions: Why was this specific network flow flagged as malicious? Which of its characteristics were anomalous? What kind of attack does it resemble? This need for transparency, trust, and actionability is the primary driver behind the burgeoning field of eXplainable Artificial Intelligence (XAI).[1]

Introducing SHAP (SHapley Additive exPlanations)

SHAP is a state-of-the-art XAI framework that provides a unified and theoretically sound approach to explaining the output of any machine learning model.[24] Its methodology is grounded in cooperative game theory, specifically the concept of Shapley values, which were developed to fairly distribute the "payout" of a collaborative game among its "players".[26] In the context of machine learning, the game is the prediction task, the payout is the model's output (e.g., the probability of an attack), and the players are the input features of the instance being explained.[26] The SHAP value for a feature represents its individual contribution to pushing the

model's prediction away from the baseline or average prediction.

Key Properties of SHAP

SHAP has gained prominence due to several desirable properties that make its explanations more reliable and consistent than many alternative methods.[24]

- **Local Accuracy:** For any single prediction, the sum of the SHAP values for each feature plus the base value equals the model's actual output for that prediction.[24] This ensures that the explanation is a faithful representation of what the model actually predicted.
- **Consistency:** This property guarantees that if a model is changed such that a feature's contribution to the prediction increases or stays the same (regardless of other features), its SHAP value will also increase or stay the same.[24] This logical property is not guaranteed by other methods, such as the built-in feature importance functions in tree-based models, which can be inconsistent and misleading.[27]
- **Global and Local Explanations:** SHAP can provide explanations at two levels. **Local explanations** detail the feature contributions for a single, specific prediction (e.g., using a force plot).[25] By aggregating these local explanations across many instances, SHAP can also provide
  global explanations that describe the model's overall behavior (e.g., using a summary plot).[25]

By integrating SHAP into the XFW-K1K2NN framework, the goal is to create a NIDS that not only achieves high performance through its sophisticated classification mechanism but also provides security analysts with the detailed, trustworthy, and actionable insights needed to effectively protect their networks.

The architecture of the XFW-K1K2NN model represents a deliberate synthesis of different machine learning paradigms. Standard kNN is a purely lazy learning algorithm, where all significant computation is deferred until a prediction is requested.[7] This approach becomes computationally prohibitive for the massive datasets common in network intrusion detection, where real-time or near-real-time performance is critical.[29] To circumvent this bottleneck, the XFW-K1K2NN framework introduces a hybrid approach. It leverages an "eager" learner—in this case, an XGBoost model—to perform a computationally intensive training phase upfront.[10] During this phase, the complex, non-linear relationships between dozens of network features are analyzed and distilled into a single, compact vector of feature importance weights. This front-loads the "learning" process. The subsequent classification step then uses the conceptually simpler, lazy kNN algorithm, but its primary weakness—the assumption of equal feature importance—has been rectified by the pre-computed weights. This hybrid design seeks to capitalize on the predictive power and feature-learning capabilities of an eager

model while retaining the localized, instance-based reasoning of a lazy model, creating a system that is both powerful and conceptually grounded.

Furthermore, the framework makes a nuanced distinction between feature importance for mechanistic use and for explanatory use. The "Feature-Weighted" (FW) component of the algorithm requires a set of quantitative values to be plugged into the weighted distance formula.[15] For this purpose, a computationally efficient metric like XGBoost's built-in 'gain' importance is sufficient and effective.[31] This is a purely mechanistic application; the weights are part of the model's internal machinery. However, research has shown that such built-in metrics can be inconsistent and unreliable when used to explain a model's behavior to a human.[27] For the "Explainable" (X) component, a more rigorous and theoretically sound method is required. SHAP provides this, offering consistent, locally accurate explanations that are far better suited for human interpretation.[25] The model's architecture is therefore sophisticated in its decoupling of these two requirements. It employs a pragmatic, efficient metric for its internal calculations and reserves a more robust, theoretically grounded metric for its external human interface, acknowledging that the optimal representation for machine computation is not always the optimal one for human cognition.

Section II: A Methodological Framework for Constructing the XFW-K1K2NN Model

This section provides a detailed, sequential blueprint for the construction and implementation of the XFW-K1K2NN model. It translates the theoretical concepts established in Section I into a concrete, four-phase engineering process. Each phase is detailed with its rationale, specific steps, and underlying logic, culminating in an architectural design that is both powerful and interpretable. The framework is designed to be modular, allowing researchers to understand and implement each component—feature weighting, K1 instance-based classification, K2 label-space refinement, and SHAP-based explainability—systematically.

2.1 Phase 1: Deriving Feature Weights via an Eager Learner (XGBoost)

The first and foundational phase of the XFW-K1K2NN model involves determining the relative importance of each feature in the NIDS dataset. This is achieved by training a high-performance, tree-based ensemble model, which acts as a "teacher" to inform the subsequent distance-based classification.

Rationale for Selecting XGBoost

Extreme Gradient Boosting (XGBoost) is selected as the eager learner for this phase for several compelling reasons. It is consistently a top-performing algorithm for supervised learning on tabular data, which is the standard format for NIDS datasets.[33] Its architecture, which sequentially builds decision trees to correct the errors of previous ones, is highly effective at capturing complex, non-linear interactions between features.[35] Furthermore, XGBoost has practical advantages for NIDS data, including its inherent ability to handle missing values and its robust regularization parameters that help prevent overfitting.[36] Most critically for this framework, trained XGBoost models provide built-in mechanisms for calculating feature importance scores, which serve as the raw material for our feature weight vector.[31]

Training and Tuning Process

The process begins by training a standard XGBClassifier on the full, preprocessed training dataset. The objective function is set for multi-class classification, where the model learns to distinguish between benign traffic and various categories of network attacks.[36] To ensure the robustness and generalizability of the resulting model, it is imperative to perform hyperparameter tuning. Key parameters to optimize include:

- n_estimators: The number of boosting rounds or trees to build.
- max_depth: The maximum depth of each individual tree, controlling model complexity.
- learning_rate (eta): A step-size shrinkage parameter used to prevent overfitting.
- subsample: The fraction of training instances to be randomly sampled for each tree.

These parameters should be tuned systematically using established techniques such as Grid Search or Randomized Search, coupled with k-fold cross-validation to obtain a reliable estimate of the model's performance on unseen data.[36]

Extracting and Normalizing Feature Weights

Once the optimal XGBoost model is trained and validated, the next step is to extract the feature importance scores. The XGBoost library provides several types of importance metrics,

including 'weight', 'cover', and 'gain'.[35] For this framework, the

'gain' importance type is recommended. 'Gain' represents the average improvement in the model's performance (i.e., the reduction in the training loss) when a particular feature is used to create a split in a tree.[35] It is a robust measure of a feature's contribution to the model's predictive power.

These raw 'gain' scores are extracted from the trained model via its feature_importances_ attribute. However, these scores are not directly suitable for use as weights in a distance metric, as their scale can be arbitrary. Therefore, they must be normalized to a consistent range. A Min-Max scaling approach is appropriate, transforming the scores to a range of or another suitable positive range. This process yields the final feature weight vector, **W**, where each element Wi reflects the normalized importance of the i-th feature.

2.2 Phase 2: Implementing the Core XFW-K1K2NN Classifier

With the feature weight vector **W** established, the core classification engine can be constructed. This phase involves implementing a modified kNN algorithm that leverages the Weighted Euclidean Distance to perform the initial, instance-based classification.

Integrating Feature Weights

The central component of the classifier is an implementation of the kNN algorithm that replaces the standard Euclidean distance with the Weighted Euclidean Distance formula, as defined in Section 1.3. This integration is the key step that transforms a standard kNN into a feature-aware classifier. The weight vector **W** derived from the XGBoost model in Phase 1 is now used to scale the squared difference for each feature in the distance calculation.

The K1 Prediction Step

The K1 prediction process follows a sequence of well-defined steps for each new, unlabeled test instance, xtest:

1. Weighted Distance Calculation: The weighted distance between xtest and every training instance, xtrain, in the dataset is computed using the formula:

$$dw(x_{test}, x_{train}) = \sum_{i=1}^{n} W_i (x_{test,i} - x_{train,i})^2$$

2. **Neighbor Identification:** The calculated distances are sorted in ascending order. The K1 training instances corresponding to the K1 smallest distances are identified. These instances constitute the set of nearest neighbors for xtest.
3. **Majority Voting:** A majority vote is conducted among the class labels of the K1 identified neighbors. In the context of multi-class NIDS, the class (i.e., the specific attack type or 'benign') that appears most frequently in this set is assigned as the initial prediction for xtest. This initial prediction is denoted as the label set LK1.[7]

Pseudocode for the K1 Step

Code snippet

```
FUNCTION K1_Predict(x_test, TrainingData, Labels, W, K1):
    distances =
    FOR EACH x_train in TrainingData:
        dist = CalculateWeightedEuclideanDistance(x_test, x_train, W)
        distances.append((dist, label_of(x_train)))

    Sort distances by distance value in ascending order

    k1_neighbors = first K1 elements of sorted distances

    // Perform majority vote
    votes = {}
    FOR EACH neighbor in k1_neighbors:
        label = neighbor.label
        votes[label] = votes.get(label, 0) + 1

    predicted_label = label with the maximum count in votes
    RETURN {predicted_label}
```

2.3 Phase 3: Adapting the K2-Step for Multi-Class Intrusion Detection

The original K2 step was designed for multi-label problems, where it identifies additional labels based on co-occurrence statistics.[18] A direct translation to the multi-class domain, where each instance has only one true label, is not possible. Therefore, this phase involves an innovative adaptation of the K2 concept, re-framing "label similarity" as "attack-type similarity" based on the patterns of confusion exhibited by a powerful classifier.

The Conceptual Bridge: From Co-occurrence to Confusion

In the multi-class NIDS context, two attack types can be considered "similar" if a sophisticated machine learning model frequently confuses one for the other. This confusion implies that the two attack types share similar characteristics in the high-dimensional feature space, making them difficult to distinguish. This provides a robust, data-driven method for quantifying the relationships between different attack categories.

Methodology for Building the Attack Similarity Matrix (ASM)

The ASM is a square matrix where each entry ASM[i, j] represents the degree to which attack type i is similar to (or confused with) attack type j. It is constructed as follows:

1. **Generate Predictions:** Use the fully trained and tuned XGBoost model from Phase 1 to make predictions on a held-out validation dataset.
2. **Construct Confusion Matrix:** Create a standard confusion matrix from these predictions, where rows represent the true labels and columns represent the predicted labels.
3. **Normalize the Matrix:** Normalize the confusion matrix by row. This is a critical step. Normalizing by row means that each cell ASM[i, j] will represent the proportion of true instances of class i that were predicted as class j. In other words, it directly calculates the conditional probability $P(\text{predicted}=j \mid \text{true}=i)$. This normalized matrix is our **Attack Similarity Matrix**. For example, if ASM = 0.3, it means that 30% of actual 'DoS Hulk' attacks were misclassified by the powerful XGBoost model as 'DoS GoldenEye', indicating a high degree of similarity.

The K2 Prediction Step

The K2 step uses the ASM to refine and enrich the initial prediction from the K1 step:

1. Let the predicted label from the K1 step be $l \in L_{K1}$.
2. Locate the row in the ASM corresponding to the true class l.
3. From this row, identify the K2 class labels (excluding l itself) that have the highest similarity scores. These are the attacks most commonly confused with the initial prediction.
4. The final output for the test instance x_{test} is an expanded set of labels, L_{final}, which includes the primary prediction from the K1 step and the top K2 similar labels. This can be presented as a ranked list: ``.

This adapted K2 step transforms the model's output from a single, definitive classification into a more nuanced and operationally valuable result. For a security analyst, this provides not just the most likely identity of a threat but also a prioritized list of alternative hypotheses to investigate. If the primary prediction is incorrect, the true attack type is likely to be among the top K2 alternatives, preventing the premature closure of an investigation and reducing the effective false negative rate for an entire family of related attacks. This makes the system more robust against subtle variations in attack vectors that might cause confusion between similar threat types.

2.4 Phase 4: Integrating the SHAP Explainer for Full Explainability

The final phase of the framework integrates the XAI component, which provides the crucial transparency needed for a NIDS. A key architectural decision is that the explainability is derived not from the kNN component but from the XGBoost model that informs it. The kNN classification, being a local, instance-based decision, is difficult to explain in a global, strategic sense. However, its behavior is fundamentally governed by the feature weights provided by the XGBoost model. By explaining the XGBoost model, we are effectively explaining the policy that drives the kNN's decisions. When SHAP explains that a certain feature was the most important factor for the XGBoost model's prediction, it directly implies that this same feature carried the most weight in the kNN's distance calculation, thereby having the greatest influence on which neighbors were selected. This proxy relationship makes an otherwise opaque part of the model (the neighbor selection process) interpretable.

Implementation

The integration of SHAP is straightforward once the XGBoost model is trained:

1. **Instantiate the Explainer:** A SHAP TreeExplainer object is created and initialized with the trained XGBoost model from Phase 1. The TreeExplainer is a highly optimized explainer specifically designed for tree-based ensemble models like XGBoost, allowing for fast and exact computation of SHAP values.[39]
   Python
   ```
   import shap
   explainer = shap.TreeExplainer(xgboost_model)
   ```

2. **Calculate SHAP Values:** For any given test instance, xtest, its corresponding SHAP values can be calculated by passing it to the explainer object. This will return a vector of the same dimension as the input features, where each value represents that feature's contribution to the prediction.
   Python
   ```
   shap_values = explainer.shap_values(x_test)
   ```

3. **Generate Visualizations:** These calculated SHAP values are then used as input to SHAP's powerful visualization functions to generate global and local explanations. These plots, including the summary plot, force plot, and dependence plot, provide rich, multi-faceted views of the model's behavior and are detailed in Section V.

This four-phase framework provides a comprehensive and reproducible methodology for constructing the XFW-K1K2NN model, resulting in a system that is not only highly accurate due to its hybrid learning architecture but also fully transparent and explainable.

Section III: Data Preparation and Engineering for Network Security

The performance of any machine learning model is fundamentally constrained by the quality and representation of the data it is trained on. In the domain of network intrusion detection, this principle is especially pertinent. NIDS datasets are characterized by high dimensionality, a mix of numerical and categorical data types, significant class imbalance, and domain-specific feature semantics. This section provides a practical guide to navigating these challenges, covering the selection of appropriate benchmark datasets, a detailed pipeline for data preprocessing, and strategies for feature engineering and selection tailored to the XFW-K1K2NN model.

3.1 Survey of Benchmark NIDS Datasets

The selection of a dataset is a foundational step in NIDS research, as it defines the context, complexity, and relevance of the experimental results. A model's performance on an older, simpler dataset may not translate to modern, real-world network environments. The choice of dataset is therefore not merely a technical decision but a strategic one that reflects the research's primary objective. For instance, a study using the well-established but dated NSL-KDD dataset is likely focused on demonstrating algorithmic novelty and ensuring direct comparability with a vast body of existing literature. In contrast, a study opting for the more complex and realistic CIC-IDS2017 dataset is signaling a focus on practical relevance, robustness, and the ability to handle near-real-world conditions. Researchers must understand that the claims they can make about their model's efficacy are directly constrained by the dataset they choose. Below is a comparative analysis of three prominent benchmark datasets.

- **NSL-KDD:** The NSL-KDD dataset was created to resolve some of the inherent statistical problems of its predecessor, the KDD'99 dataset, primarily by removing a large number of redundant and duplicate records from the training and test sets.[41] It consists of 41 features per connection record and is labeled with a specific attack type or as 'normal'.[42] The attacks are broadly categorized into four main types: Denial of Service (DoS), Probing (Probe), User to Root (U2R), and Remote to Local (R2L).[44] While its improved structure makes it a better choice than KDD'99 for benchmarking, the dataset is now over a decade old, and the traffic patterns and attack vectors it contains may not accurately represent the complexities of modern networks.[46]
- **UNSW-NB15:** Developed by the Australian Centre for Cyber Security (ACCS) in 2015, the UNSW-NB15 dataset was explicitly designed to address the limitations of older datasets like KDD'99.[30] It was generated using the IXIA PerfectStorm tool to create a hybrid of real modern normal network activities and synthetic contemporary attack behaviors.[47] The dataset features 49 attributes and includes nine modern attack categories, such as Fuzzers, Analysis, Backdoors, Exploits, and Shellcode, which are more representative of the current threat landscape.[47]
- **CIC-IDS2017:** This is one of the most recent and widely-cited benchmark datasets, created by the Canadian Institute for Cybersecurity (CIC).[29] It was generated over a five-day period and features a comprehensive network topology and realistic background traffic generated by profiling the abstract behavior of 25 users across protocols like HTTP, HTTPS, FTP, and SSH.[46] It contains the most up-to-date common attacks, including Brute Force (FTP and SSH), Heartbleed, Web Attacks (Brute Force, XSS, SQL Injection), Infiltration, Botnet, and various DoS/DDoS attacks.[46] The data was processed using the CICFlowMeter tool, which extracted over 80 network flow features.[46] Its primary strengths are its realism, diversity of attacks, and large volume. However, it also presents significant challenges, including severe class imbalance and identified data quality issues

like missing labels and duplicate columns that require careful preprocessing.[20]

Table 1: Comparison of Benchmark NIDS Datasets

Dataset Name	Year Released	Total Instances	Number of Features	Attack Categories	Benign/ Attack Ratio (Approx.)	Key Strengths	Known Limitations
NSL-KDD	2009	~148,517	41	DoS, Probe, U2R, R2L	45% / 55%	Standardized benchmark, removed KDD'99 duplicates, widely used for comparability.	Dated attack vectors, does not reflect modern network traffic complexity.[30]
UNSW-NB15	2015	~2,540,044	49	Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode,	87% / 13%	Hybrid of real and synthetic traffic, modern attack types, larger scale than NSL-KDD.	Significant class imbalance, some redundancy in features.[30]

				Worms			
CIC-IDS2017	2017	~2,830,000	78+	Brute Force, Heartbleed, Botnet, DoS, DDoS, Web Attacks, Infiltration	80% / 20%	Realistic background traffic, diverse and up-to-date attacks, high volume, detailed flow features.	Severe class imbalance, known data quality issues (missing values, duplicates), very large file sizes.[20]

3.2 The Data Preprocessing Pipeline

Raw NIDS datasets are not suitable for direct input into machine learning models. A rigorous preprocessing pipeline is essential to clean, transform, and structure the data appropriately. The choices made during this stage are not neutral; they encode critical assumptions about the data and fundamentally shape the feature space in which a distance-based algorithm like kNN operates. A poor choice can distort the geometry of the data, making it impossible for the algorithm to identify meaningful patterns.

Data Cleaning

The first step is to address data quality issues. This involves:

- **Handling Missing and Infinite Values:** Datasets like CIC-IDS2017 are known to contain missing information and may also have Infinity or NaN (Not a Number) values, often resulting from calculations like packets-per-second for flows with zero duration.[29] These must be handled, for example, by imputing them with the mean, median, or a constant

value like zero, or by removing the affected rows if they are few in number.
- **Removing Duplicates:** Some datasets may contain duplicate columns or rows, which can bias the model and should be removed.[29]

Feature Scaling and Normalization

This is arguably the most critical preprocessing step for any distance-based algorithm, including the XFW-K1K2NN model. NIDS datasets contain features with vastly different scales and ranges—for example, src_bytes can range from zero to millions, while a rate feature like serror_rate ranges from 0 to 1.[6] Without scaling, the features with larger magnitudes will completely dominate the Euclidean distance calculation, effectively rendering features with smaller ranges invisible to the algorithm.[8] The choice of scaler encodes an assumption about the data's distribution:

- **StandardScaler:** This scaler transforms features to have a mean of 0 and a standard deviation of 1. It assumes that the features are normally distributed and that the magnitude of deviation from the mean is the most important aspect of a feature's value.
- **MinMaxScaler:** This scaler transforms features to a specific range, typically . It is less sensitive to outliers than StandardScaler and assumes that the absolute position of a value within its observed min-max range is what matters most.

Given the non-standard distributions of many network features, MinMaxScaler is often a robust choice. All numerical features must be scaled before being used in the model.

Encoding Categorical Features

NIDS datasets contain nominal categorical features that are highly informative, such as protocol_type ('tcp', 'udp', 'icmp'), service, and flag.[45] These must be converted into a numerical format.

- **Label Encoding:** Assigns a unique integer to each category. This is memory-efficient but should be used with caution for kNN, as it imposes an artificial ordinal relationship (e.g., 'udp'=2 is "greater than" 'tcp'=1) that can distort distance calculations.
- **One-Hot Encoding:** Creates a new binary feature for each category. This avoids imposing an artificial order and is generally the preferred method for kNN, as it treats each category as an independent entity.[53] However, it can lead to a very high-dimensional feature space if a categorical variable has many unique values (high

cardinality).

Table 2: Selected Feature Descriptions for the CIC-IDS2017 Dataset

Feature Name	Data Type	Description
Destination Port	Numerical	The port number of the destination system.
Flow Duration	Numerical	The duration of the flow in microseconds.
Total Fwd Packets	Numerical	Total packets in the forward direction.
Total Backward Packets	Numerical	Total packets in the backward direction.
Total Length of Fwd Packets	Numerical	Total size of packets in the forward direction.
Fwd Packet Length Max	Numerical	Maximum size of a packet in the forward direction.
Fwd Packet Length Mean	Numerical	Mean size of a packet in the forward direction.
Flow IAT Mean	Numerical	Mean time between two packets sent in the flow.
Fwd IAT Mean	Numerical	Mean time between two packets sent in the forward direction.
Fwd PSH Flags	Categorical (Binary)	Number of packets with PSH flag set in the forward direction (0 for no, 1 for yes).

Fwd URG Flags	Categorical (Binary)	Number of packets with URG flag set in the forward direction (0 for no, 1 for yes).
Fwd Header Length	Numerical	Total bytes used for headers in the forward direction.
Fwd Packets/s	Numerical	Number of forward packets per second.
Packet Length Mean	Numerical	Mean length of a packet.
Packet Length Std	Numerical	Standard deviation of packet length.
FIN Flag Count	Numerical	Number of packets with FIN flag.
SYN Flag Count	Numerical	Number of packets with SYN flag.
RST Flag Count	Numerical	Number of packets with RST flag.
PSH Flag Count	Numerical	Number of packets with PSH flag.
ACK Flag Count	Numerical	Number of packets with ACK flag.
Down/Up Ratio	Numerical	The ratio of the number of backward packets to forward packets.
Average Packet Size	Numerical	The average size of a packet.

Subflow Fwd Bytes	Numerical	The average number of bytes in a sub-flow in the forward direction.
Init_Win_bytes_forward	Numerical	The total number of bytes sent in the initial window in the forward direction.
Active Mean	Numerical	Mean time the flow was active before becoming idle.
Idle Mean	Numerical	Mean time the flow was idle before becoming active.
Label	Categorical	The label of the network flow (Benign or specific attack type).

(Note: This is a representative subset. The full CIC-IDS2017 dataset contains over 78 features. A complete data dictionary is essential for any research project.) [46]

3.3 Feature Selection and Engineering

Given the high dimensionality of modern NIDS datasets, feature selection is a crucial step to improve model performance, reduce computational overhead, and enhance interpretability.[20]

- **Filter Methods:** These methods assess the relevance of features by their correlation with the target variable or other statistical properties, independent of any machine learning model. Techniques like correlation analysis can identify and remove redundant features, while methods like Mutual Information can rank features based on their relationship with the class labels.[20]
- **Wrapper Methods:** These methods use a specific machine learning model to evaluate the utility of different subsets of features. Recursive Feature Elimination (RFE), for example, iteratively trains a model and removes the least important feature until the desired number of features is reached.[53]
- **Embedded Methods:** These methods perform feature selection as part of the model

training process itself. This is the approach most aligned with the XFW-K1K2NN framework. The feature importance scores ('gain') generated by the trained XGBoost model in Phase 1 can be used directly to select the top N most informative features for the subsequent kNN classification step.[31] This ensures that the features used for classification are the same ones that the powerful XGBoost model found most discriminative.

- **Domain-Specific Engineering:** Beyond selecting from existing features, creating new, more informative features can significantly boost performance. This requires domain knowledge of network protocols and attack behaviors.[57] For example, one could engineer features that capture the ratio of forward to backward packets, the proportion of packets with specific flags (e.g., SYN, FIN), or statistical properties of packet inter-arrival times over different window sizes. This process transforms raw data into higher-level representations that may better capture the subtle signatures of malicious activity.[34]

By following this comprehensive data preparation and engineering workflow, researchers can create a clean, well-structured, and informative feature set that maximizes the potential of the XFW-K1K2NN model to accurately and robustly detect network intrusions.

Section IV: A Practical Guide to Application, Validation, and Performance Evaluation

After preparing the data and constructing the model architecture, the next critical phase involves training, tuning, and rigorously evaluating the XFW-K1K2NN model. This section provides a comprehensive guide to this process, with a strong emphasis on methodologies appropriate for the unique challenges of the network security domain, particularly the pervasive issue of class imbalance. It covers hyperparameter optimization, the selection of meaningful evaluation metrics, performance visualization techniques, and the importance of establishing a robust performance baseline.

4.1 Model Training and Hyperparameter Tuning

The performance of the XFW-K1K2NN model is sensitive to the choice of its key hyperparameters. These parameters are not universal and must be optimized for the specific characteristics of the dataset being used.[7]

- **Key Hyperparameters:**

    - **K1:** The number of nearest neighbors to consider in the initial instance-space search. This is the most critical parameter for the kNN component.
    - **K2:** The number of similar attack labels to append in the second, label-space search step.
    - **XGBoost Parameters:** The hyperparameters of the XGBoost model used for feature weighting (e.g., n_estimators, max_depth, learning_rate) must also be tuned, as described in Section 2.1. The quality of the feature weights is dependent on the performance of this underlying model.
- Tuning Strategy and the Bias-Variance Trade-off:
  The process of tuning K1 is a classic exercise in managing the bias-variance trade-off.7 A very small value for
  K1 (e.g., K1=1) creates a model with low bias but high variance. The decision boundary will be highly complex and sensitive to individual data points, making the model prone to overfitting and susceptible to noise and outliers.[9] Conversely, a very large value for
  K1 creates a model with high bias and low variance. The decision boundary becomes overly smooth, potentially ignoring important local patterns in the data and leading to underfitting.[9]
  The optimal value for K1 (and similarly for K2) must be found empirically. The recommended approach is to use k-fold cross-validation on the training set. A range of values for the hyperparameter is tested, and for each value, the model is trained and evaluated multiple times on different splits of the data. The value that yields the best average performance on the validation folds is selected. A common visualization technique is the "elbow method," where a performance metric (such as Macro F1-Score) is plotted against different values of K. The optimal K is often found at the "elbow" of the curve, the point where increasing K further yields diminishing returns in performance improvement.[12]

4.2 Evaluating NIDS Performance in Imbalanced Environments

The most common pitfall in evaluating NIDS models is the misuse of inappropriate performance metrics. The highly imbalanced nature of network traffic data renders many standard metrics, particularly accuracy, not just uninformative but dangerously misleading.

The Failure of Accuracy

In a typical NIDS dataset, benign traffic constitutes the vast majority of instances, often

exceeding 99% of the data, while malicious traffic from various attack classes makes up a tiny minority.[20] In such a scenario, a trivial classifier that simply predicts "benign" for every single instance would achieve 99% accuracy. While numerically high, this model is operationally useless because it fails to detect a single attack, completely defeating the purpose of a NIDS.[59] This phenomenon, known as the accuracy paradox, highlights why accuracy must be avoided as a primary evaluation metric for imbalanced classification problems.[59]

The Confusion Matrix as the Source of Truth

All meaningful classification metrics for NIDS are derived from the confusion matrix, which provides a detailed breakdown of a model's predictions against the ground truth. For a binary problem (Attack vs. Benign), it consists of four values:

- **True Positives (TP):** Malicious instances correctly identified as malicious.
- **True Negatives (TN):** Benign instances correctly identified as benign.
- **False Positives (FP):** Benign instances incorrectly identified as malicious (a false alarm).
- False Negatives (FN): Malicious instances incorrectly identified as benign (a missed attack).

60

Essential Metrics for NIDS

The following metrics provide a nuanced and reliable assessment of a NIDS's performance by focusing on the trade-offs between detecting attacks and generating false alarms.

- **Precision (Positive Predictive Value):** Calculated as TP/(TP+FP), precision answers the question: "Of all the instances we flagged as attacks, what proportion were actually attacks?".[62] High precision is vital for operational efficiency. Low precision means a high number of false positives, which leads to "alert fatigue" for security analysts, causing them to ignore or distrust the system's outputs.[64]
- **Recall (Sensitivity or True Positive Rate):** Calculated as TP/(TP+FN), recall answers the question: "Of all the actual attacks that occurred, what proportion did we successfully detect?".[62] High recall is paramount for security effectiveness. Low recall means a high number of false negatives (missed attacks), which can lead to catastrophic security breaches.[64]
- **F1-Score:** The F1-Score is the harmonic mean of precision and recall, calculated as

2×(Precision×Recall)/(Precision+Recall).[61] It provides a single, balanced measure of a model's performance, making it one of the most important and widely used metrics for imbalanced classification. It is high only when both precision and recall are high.

The Importance of Macro-Averaging in Multi-Class NIDS

When dealing with multiple attack classes, these metrics must be aggregated. The choice between averaging methods is not merely technical but reflects a core philosophical stance on the security problem.

- **Micro-Averaging:** Aggregates the TPs, FPs, and FNs from all classes before calculating the metric. This method gives equal weight to each individual instance, meaning the final score will be dominated by the model's performance on the largest classes (typically the 'benign' class).[66] This can hide poor performance on rare but critical attack types.
- **Macro-Averaging:** Calculates the metric independently for each class and then takes the unweighted average of the per-class scores. This method gives equal weight to each class, regardless of its size.[66] For NIDS, this is almost always the more meaningful approach. It ensures that the model's ability to detect a rare 'Worm' attack contributes just as much to the final score as its ability to classify common 'Benign' traffic. It directly measures the model's robustness against a *diversity* of threats, which is the ultimate goal of a comprehensive NIDS.

Table 3: Evaluation Metrics for Imbalanced NIDS Classification

Metric Name	Formula	Interpretation in NIDS Context	Best for...
Accuracy	(TP+TN)/(TP+TN+FP+FN)	**Misleading.** High value can be achieved by ignoring all attacks. Should not be used as a primary metric.	Balanced datasets only.
Precision	TP/(TP+FP)	Measures the reliability of alerts. High precision minimizes false alarms and analyst	Assessing the operational cost of false positives.

		alert fatigue.	
Recall	TP/(TP+FN)	Measures the attack detection rate. High recall minimizes missed attacks and security breaches.	Assessing the core security effectiveness of the system.
F1-Score	2×Precision+Recall Precision×Recall	The harmonic mean of Precision and Recall. Provides a single score balancing the two concerns.	A primary metric for overall model performance on a single class.
Macro-Avg F1	C1Σi=1CF1i	Averages the F1-score across all C classes, giving each class equal weight.	The **most recommended** primary metric for multi-class, imbalanced NIDS evaluation.
ROC AUC	Area Under the ROC Curve	Measures the model's ability to discriminate between classes across all thresholds. Can be optimistic on imbalanced data.	Comparing general discriminative power of models.
PR AUC	Area Under the Precision-Recall Curve	Measures the trade-off between precision and recall. More informative than ROC AUC for highly imbalanced data.	Evaluating performance on the minority (attack) class.

4.3 Visualizing Performance: ROC and Precision-Recall Curves

Beyond single-number metrics, performance curves provide a more holistic view of a model's behavior across different decision thresholds.

- **ROC (Receiver Operating Characteristic) Curve:** This curve plots the True Positive Rate (Recall) against the False Positive Rate (FP/(FP+TN)).[68] The Area Under the Curve (ROC AUC) is a measure of the model's ability to distinguish between classes. A perfect classifier has an AUC of 1.0, while a random guess has an AUC of 0.5. However, because the False Positive Rate is calculated using the large number of true negatives, the ROC curve can be overly optimistic in highly imbalanced scenarios.[59]
- **PR (Precision-Recall) Curve:** This curve plots Precision against Recall for different thresholds.[62] For imbalanced datasets, the PR curve is often more informative as both its axes are focused on the performance of the minority (positive) class.[59] A skilled model will have a curve that stays high and to the right, and the Area Under the PR Curve (PR AUC) serves as a robust summary metric.

4.4 Establishing a Performance Baseline

To credibly demonstrate the value of the proposed XFW-K1K2NN model, its performance must be contextualized. This requires comparing it against a set of well-chosen baseline models on the same preprocessed data and using the same evaluation metrics. A strong set of baselines should include:

1. **Standard kNN:** This model will use the same data but without feature weighting or the K2 step. Comparing against this baseline isolates the performance contribution of the novel components of the XFW-K1K2NN framework.
2. **The Standalone XGBoost Model:** This is the powerful eager learner used in Phase 1 to generate feature weights. It is crucial to show that the full hybrid XFW-K1K2NN model provides a performance benefit over simply using the XGBoost classifier on its own.
3. **Other State-of-the-Art Classifiers:** Including other commonly used NIDS models like Random Forest, Support Vector Machines, or a simple Multi-Layer Perceptron (MLP) provides a broader benchmark against the current state of the art in the field.[33]

By conducting a rigorous, multi-faceted evaluation using appropriate metrics and strong baselines, researchers can generate credible and meaningful evidence of the XFW-K1K2NN

model's effectiveness in the challenging domain of network intrusion detection.

Section V: Generating and Interpreting Actionable Explanations

The defining feature of the XFW-K1K2NN model is its "X" component: the integration of explainability. While high performance on metrics like the Macro F1-Score is essential, the ability to translate a model's decisions into human-understandable insights is what makes it a truly valuable tool for a security operations center (SOC). This section details how to leverage the integrated SHAP explainer to move beyond quantitative evaluation and generate actionable intelligence. It covers the generation and interpretation of global explanations, which reveal the model's overall detection strategy, and local explanations, which diagnose individual security alerts.

5.1 Global Explanations: Understanding the Model's Overall Strategy

Global explanations provide a high-level overview of the model's behavior across the entire dataset. They answer questions like: "Which features are most important overall?" and "What is the general relationship between feature values and the likelihood of an attack?"

SHAP Summary Plot (Beeswarm Plot)

The SHAP summary plot is the most powerful tool for global model explanation, offering a rich, dense visualization of feature importance and effects.[40] It is constructed by plotting the SHAP value of every feature for every sample in the dataset.

- **Feature Importance:** Features are ranked on the y-axis by their overall importance, which is calculated as the sum of the absolute SHAP values across all samples. The most important features appear at the top.[25]
- **Impact on Prediction:** The x-axis represents the SHAP value. A positive SHAP value indicates that the feature's value for that instance pushed the model's prediction towards a higher probability of being an attack. A negative SHAP value pushed the prediction towards "benign".[36]

- **Feature Value Correlation:** The color of each point represents the original value of that feature for that instance, from low (blue) to high (red).[26]

By combining these three elements, an analyst can quickly discern complex patterns. For example, a summary plot might reveal that the Flow Duration feature is highly important. Furthermore, it might show a clear pattern where high values of Flow Duration (red dots) are consistently associated with positive SHAP values, indicating that long-lasting network flows are strong indicators of an attack in this model's logic. Conversely, it might show that for the Destination Port feature, only a few specific low values (blue dots) have a strong positive impact, perhaps corresponding to known vulnerable ports.

SHAP Bar Plot

For a simpler, more direct view of feature importance, a SHAP bar plot can be used. This chart displays the mean absolute SHAP value for each feature, averaged across all samples.[25] This provides a clear and unambiguous ranking of the features that have the most influence on the model's predictions, on average. It is crucial to note that this method of determining global importance is considered more reliable and consistent than using the native feature importance functions provided by XGBoost (e.g., 'weight', 'cover', 'gain'), which can sometimes produce contradictory rankings and are biased towards certain feature types or positions within the trees.[27]

5.2 Local Explanations: Diagnosing Individual Alerts

While global explanations are useful for understanding the model, local explanations are the key to making it actionable in a real-time security context. They explain *why a single, specific prediction was made*.

SHAP Force Plot

The force plot is a compelling visualization for explaining an individual prediction.[25] It represents features as "forces" that push the prediction away from the base value (the average prediction over the entire dataset).

- **Red Features:** Features shown in red had values that increased the prediction score (pushed it towards "attack"). The length of the red bar corresponds to the magnitude of that feature's impact.
- Blue Features: Features shown in blue had values that decreased the prediction score (pushed it towards "benign").
  The plot dynamically shows how these competing forces balance out to arrive at the final prediction score for that specific instance.26 For a SOC analyst triaging an alert, this is invaluable. It immediately highlights the top 3-5 most anomalous characteristics of the network flow that caused it to be flagged, allowing the analyst to focus their investigation instantly.

SHAP Waterfall Plot

The waterfall plot provides a more detailed, static view of a single prediction's explanation.[25] It starts with the base value at the bottom and shows how the positive and negative SHAP contributions of each feature are sequentially added to build up to the final prediction value at the top. This provides a clear, step-by-step accounting of how the model arrived at its decision for a single instance.

The utility of these explanations extends far beyond simply validating a single alert. They can be used proactively to generate new security intelligence. For example, a global SHAP summary plot might reveal that a seemingly benign feature, such as the mean idle time between packets (Idle Mean), is a top predictor for a specific type of Botnet. This is not just a statistical artifact; it is an empirical discovery that this Botnet's command-and-control (C2) traffic exhibits a distinctive "heartbeat" or timing signature. An astute analyst can leverage this insight to form a new, testable hypothesis: "This family of malware can be identified by its characteristic C2 communication pattern with an idle time between X and Y seconds." This hypothesis can then be used to write new detection rules for other security tools (like Snort or Suricata) or to perform proactive "threat hunting" by querying historical network logs for this specific pattern. In this way, the XAI component transforms the NIDS from a reactive detection system into a proactive discovery engine.

5.3 Uncovering Deeper Insights: Dependence and Interaction Plots

For more advanced analysis, SHAP provides tools to explore non-linear relationships and feature interactions.

SHAP Dependence Plot

A dependence plot shows the relationship between the value of a single feature (on the x-axis) and its corresponding SHAP value (on the y-axis) for every instance in the dataset.[39] This can reveal complex, non-linear effects that a simple bar chart of importance would miss. For example, the plot might show that the

Packet Length Mean has a SHAP value near zero for most of its range, but its impact on the prediction score increases exponentially after it crosses a certain threshold, a pattern often associated with specific exploit payloads or data exfiltration techniques.

A powerful feature of the dependence plot is its ability to automatically color the points based on the value of a second feature that has the strongest interaction effect with the feature being plotted. This vertical dispersion of colors can reveal interaction effects. For instance, a dependence plot for Source Port might show a wide vertical spread of colors corresponding to the Destination Port. This would indicate that the importance of a specific source port depends heavily on the destination port it is communicating with, a classic pattern in network reconnaissance and exploitation.

SHAP Interaction Values

For the most in-depth analysis, SHAP can compute not just the main effect of each feature but also the pairwise interaction effects.[39] The SHAP interaction value between two features measures the additional impact on the prediction that comes from knowing both of their values, beyond the sum of their individual impacts. Analyzing these interactions can help uncover sophisticated attack patterns that rely on a combination of several seemingly normal network characteristics to succeed.

These local and global explanations also create a powerful feedback loop for model improvement. When the model produces a misclassification (either a false positive or a false negative), the SHAP explanation for that incorrect prediction becomes a valuable diagnostic tool. For instance, if a benign flow from a new video conferencing application is flagged as a DoS attack (a false positive), the force plot might reveal that the feature Fwd Packets/s was the primary culprit. The analyst can investigate and determine that the high packet rate of the video stream mimics a DoS attack. This provides two clear, actionable paths for improving the model. First, this misclassified instance, now correctly labeled, can be added to the training set to help the model learn to distinguish this specific benign pattern. Second, it suggests a

need for more sophisticated feature engineering. The feature Fwd Packets/s alone is insufficient. A new, engineered feature, such as the ratio of forward packets to total payload size, might be necessary to differentiate between high-volume, high-payload data transfers (benign streaming) and high-volume, low-payload flooding attacks (malicious DoS). In this way, the XAI output directly informs and guides the next iteration of the data science lifecycle, leading to a more robust and accurate model over time.

Section VI: Advanced Considerations and Future Research Directions

While the XFW-K1K2NN framework presents a powerful and interpretable approach to network intrusion detection, its practical deployment and future evolution require careful consideration of its inherent limitations and potential avenues for enhancement. This final section addresses the critical aspects of computational complexity and scalability, the challenges of deploying such a model in a real-time environment, and outlines promising directions for future research that could extend and improve upon the proposed architecture.

6.1 Computational Complexity and Scalability Analysis

The architectural design of XFW-K1K2NN, which combines an eager learner with a lazy learner, introduces specific performance characteristics that must be understood.

The Lazy Learning Bottleneck

The primary computational bottleneck of the XFW-K1K2NN model lies in the K1 prediction step. As a derivative of the kNN algorithm, it inherits kNN's "lazy" nature, meaning that for every single test instance, it must compute the weighted distance to *every* instance in the stored training set.[3] For a training set with

N samples and D features, the time complexity of this brute-force search is O(N×D). While manageable for small datasets, this becomes computationally prohibitive for modern NIDS datasets, which can contain millions of instances, making the model too slow for real-time

applications.[7]

Optimization with Approximate Nearest Neighbor (ANN) Search

To overcome this scalability issue, the brute-force search for nearest neighbors can be replaced with more sophisticated, optimized search algorithms. These methods, often grouped under the umbrella of Approximate Nearest Neighbor (ANN) search, build specialized data structures during the training phase to dramatically accelerate the search process at inference time. Common techniques include:

- **KD-Trees:** These are space-partitioning data structures that recursively partition the feature space, allowing for efficient pruning of the search space. They are most effective for low-dimensional data (typically D<20).[71]
- **Ball Trees:** These structures partition data within a series of nested hyperspheres. They are generally more robust than KD-Trees for higher-dimensional data.[71]

By using these structures, the query time for finding the nearest neighbors can be reduced from O(N×D) to approximately O(D×logN).[71] This optimization is crucial for making the XFW-K1K2NN model viable for larger datasets. Popular machine learning libraries like scikit-learn provide efficient implementations of these algorithms, which can be easily integrated by setting the

algorithm parameter in a kNN classifier to 'kd_tree' or 'ball_tree'.[71]

6.2 Challenges of Real-Time Deployment

Moving from a research environment to a live, operational network introduces further challenges beyond raw computational speed.

- **Inference Latency:** Even with ANN optimization, the time required to preprocess a network flow, extract its features, and perform the K1/K2 classification and SHAP explanation must be low enough to meet the demands of a real-time NIDS. High latency could mean that an attack is detected only after it has already caused damage. The entire pipeline must be optimized for low-latency inference.
- **Concept Drift:** Network environments are dynamic. The statistical properties of network traffic change over time due to the introduction of new applications, changes in user behavior, and the emergence of novel attack techniques. This phenomenon, known as "concept drift," will cause the performance of any model trained on a static, historical

dataset to degrade over time. To remain effective, a deployed NIDS must have a strategy for dealing with concept drift, such as periodic retraining on new data or the implementation of online learning mechanisms that can adapt the model incrementally without requiring a full retraining cycle.

The architecture of XFW-K1K2NN is a carefully constructed series of trade-offs designed to balance competing objectives. The inherent simplicity and intuitiveness of the core kNN algorithm are traded for the superior predictive power of the XGBoost model, which is needed to handle the complexity of NIDS data. This introduces opacity, which is then mitigated by adding the SHAP framework—a computationally expensive layer—to restore transparency. The primary performance bottleneck of the kNN component, its slow brute-force search, can be addressed by using approximate search algorithms, which trades a degree of exactness for a significant gain in speed. This chain of compromises highlights a fundamental truth in applied machine learning: designing a practical, effective, and trustworthy NIDS is not about finding a single "best" algorithm. Rather, it is an engineering discipline focused on intelligently navigating these trade-offs to build a system that is sufficiently performant, scalable, and interpretable for its specific operational context.

6.3 Future Research Directions

The XFW-K1K2NN framework provides a solid foundation that can be extended and improved in several promising research directions.

- **Dynamic and Localized Feature Weighting:** The current model computes a single, global vector of feature weights. A more advanced implementation could develop a dynamic weighting scheme. For example, an attention mechanism could be employed to learn weights that are specific to the local neighborhood of the test instance, allowing the model to focus on different features for different types of traffic.
- **Unsupervised and Semi-Supervised Adaptation:** The current framework is fully supervised and requires a large, labeled dataset. Adapting the model for unsupervised anomaly detection, where attack labels are scarce or unavailable, is a critical area for future work. This could involve using clustering algorithms to define "normal" neighborhoods in the feature space and then identifying instances that are distant from all known clusters as anomalies.
- **Graph-Based Attack Similarity:** The current K2 step uses a confusion matrix to model attack similarity. A more sophisticated approach could represent attack classes as nodes in a graph. A Graph Neural Network (GNN) could then be trained to learn rich embeddings for each attack type based on their feature-space relationships, potentially capturing more complex similarities than a simple confusion matrix.
- **Integration of Advanced XAI Techniques:** The explainability component could be

enhanced by incorporating other XAI methods alongside SHAP. For example, providing **counterfactual explanations** (e.g., "This flow would have been classified as 'benign' if its Flow Duration had been 500ms shorter") can be highly intuitive for security analysts, as it provides a clear, minimal change that would alter the outcome, offering a different and complementary perspective to SHAP's feature contribution scores.

By addressing these challenges and exploring these future directions, the principles underlying the XFW-K1K2NN model can continue to evolve, contributing to the development of next-generation Network Intrusion Detection Systems that are more accurate, scalable, and fundamentally more trustworthy.

Conclusion

The Explainable Feature-Weighted K1K2NN (XFW-K1K2NN) framework presented in this guide offers a robust and novel solution to the dual challenges of performance and interpretability in network intrusion detection. By systematically synthesizing concepts from lazy learning, ensemble modeling, and explainable AI, the proposed architecture provides a comprehensive methodology for researchers and practitioners aiming to build more effective and trustworthy NIDS.

The model's hybrid eager/lazy learning paradigm addresses the primary weaknesses of its constituent parts, using the predictive power of XGBoost to overcome the feature-agnostic nature of kNN, while retaining the latter's intuitive, instance-based reasoning. The adaptation of the K1K2NN's two-stage classification process from a multi-label to a multi-class context, using a confusion-based Attack Similarity Matrix, transforms the model's output from a simple prediction into a prioritized list for alert triage, significantly enhancing its operational value.

Crucially, the integration of SHAP for explainability elevates the framework beyond a mere classification tool. It establishes a clear distinction between the mechanistic use of feature importance for internal model calculations and the rigorous, theoretically-grounded use of SHAP for human-centric explanation. The detailed local and global explanations generated by SHAP not only provide the transparency required to validate and trust individual alerts but also create a powerful feedback loop. These explanations serve as a diagnostic tool for identifying model weaknesses and as a discovery engine for generating new security hypotheses, directly informing both model retraining and proactive threat hunting efforts.

The practical implementation of this framework requires careful attention to the nuances of NIDS data, including rigorous preprocessing, appropriate feature scaling, and the selection of evaluation metrics, like the Macro-Averaged F1-Score, that are robust to severe class imbalance. While computational scalability remains a challenge, it can be effectively mitigated

through the use of approximate nearest neighbor search algorithms.

Ultimately, the XFW-K1K2NN model represents a deliberate navigation of the inherent trade-offs between performance, complexity, and transparency. It serves as a blueprint for a class of systems that do not treat explainability as an afterthought but rather as a core design principle, paving the way for next-generation NIDS that are not only more accurate but also function as collaborative partners with human security analysts in the complex and dynamic task of defending network infrastructures. Future research building upon this foundation—exploring dynamic weighting, unsupervised adaptations, and more advanced explainability techniques—will continue to push the boundaries of what is possible in the field of intelligent and transparent cybersecurity.

Works cited

1. (PDF) A Survey on Explainable Artificial Intelligence for Internet ..., accessed August 17, 2025, https://www.researchgate.net/publication/385985733_A_Survey_on_Explainable_Artificial_Intelligence_for_Internet_Traffic_Classification_and_Prediction_and_Intrusion_Detection
2. (PDF) Robust Network Intrusion Detection Through Explainable ..., accessed August 17, 2025, https://www.researchgate.net/publication/361638658_Robust_Network_Intrusion_Detection_through_Explainable_Artificial_Intelligence_XAI
3. k-nearest neighbors algorithm - Wikipedia, accessed August 17, 2025, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
4. Introduction to machine learning: k-nearest neighbors - PMC - PubMed Central, accessed August 17, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC4916348/
5. K-nearest Neighbors | Brilliant Math & Science Wiki, accessed August 17, 2025, https://brilliant.org/wiki/k-nearest-neighbors/
6. K Nearest Neighbor : Step by Step Tutorial - ListenData, accessed August 17, 2025, https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html
7. What is the k-nearest neighbors algorithm? - IBM, accessed August 17, 2025, https://www.ibm.com/think/topics/knn
8. K-Nearest Neighbors (KNN) Classification with scikit-learn - DataCamp, accessed August 17, 2025, https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
9. Wikipedia K Nearest Neighbor Algorithm | PDF | Formalism (Deductive) - Scribd, accessed August 17, 2025, https://www.scribd.com/document/72546572/Wikipedia-k-Nearest-Neighbor-Algorithm
10. What is eager learning and lazy learning? - Artificial Intelligence Stack Exchange, accessed August 17, 2025, https://ai.stackexchange.com/questions/22769/what-is-eager-learning-and-lazy-l

earning
11. High-Level K-Nearest Neighbors (HLKNN): A Supervised Machine Learning Model for Classification Analysis - MDPI, accessed August 17, 2025, https://www.mdpi.com/2079-9292/12/18/3828
12. K-Nearest Neighbor (KNN) Explained - Pinecone, accessed August 17, 2025, https://www.pinecone.io/learn/k-nearest-neighbor/
13. Understanding Euclidean Distance: From Theory to Practice - DataCamp, accessed August 17, 2025, https://www.datacamp.com/tutorial/euclidean-distance
14. Different Types of Distances Used in Machine Learning | by Srijani Chaudhury - Medium, accessed August 17, 2025, https://medium.com/swlh/different-types-of-distances-used-in-machine-learning-ec7087616442
15. Mastering Euclidean Distance in ML - Number Analytics, accessed August 17, 2025, https://www.numberanalytics.com/blog/ultimate-guide-euclidean-distance-machine-learning
16. When would one use Manhattan distance as opposed to Euclidean distance?, accessed August 17, 2025, https://datascience.stackexchange.com/questions/20075/when-would-one-use-manhattan-distance-as-opposed-to-euclidean-distance
17. A Complete Guide to K-Nearest Neighbors Algorithm – KNN using Python, accessed August 17, 2025, https://ashutoshtripathi.com/2019/08/05/a-complete-guide-to-k-nearest-neighbors-algorithm-knn-using-python/
18. K1K2NN: A Novel Multi-Label Classification Approach Based on Neighbours to Predict Covid Drug Side Effects from Chemical Properties - ResearchGate, accessed August 17, 2025, https://www.researchgate.net/publication/369276680_K1K2NN_A_Novel_Multi-Label_Classification_Approach_Based_on_Neighbours_to_Predict_Covid_Drug_Side_Effects_from_Chemical_Properties?_share=1
19. K1K2NN: A novel multi-label classification approach based on neighbors for predicting COVID-19 drug side effects - PubMed, accessed August 17, 2025, https://pubmed.ncbi.nlm.nih.gov/38579549/
20. Important Features of CICIDS-2017 Dataset For Anomaly Detection in High Dimension and Imbalanced Class Dataset - Sriwijaya University Repository, accessed August 17, 2025, https://repository.unsri.ac.id/59517/1/Article-IJEEI-Important%20Features%20of%20CICIDS-2017%20Dataset.pdf
21. List of features in NSL-KDD dataset | Download Table - ResearchGate, accessed August 17, 2025, https://www.researchgate.net/figure/List-of-features-in-NSL-KDD-dataset_tbl1_278406516
22. Weighted K-NN - GeeksforGeeks, accessed August 17, 2025, https://www.geeksforgeeks.org/machine-learning/weighted-k-nn/

23. [2503.02630] Weighted Euclidean Distance Matrices over Mixed Continuous and Categorical Inputs for Gaussian Process Models - arXiv, accessed August 17, 2025, https://arxiv.org/abs/2503.02630
24. 18 SHAP – Interpretable Machine Learning, accessed August 17, 2025, https://christophm.github.io/interpretable-ml-book/shap.html
25. SHAP : A Comprehensive Guide to SHapley Additive exPlanations - GeeksforGeeks, accessed August 17, 2025, https://www.geeksforgeeks.org/machine-learning/shap-a-comprehensive-guide-to-shapley-additive-explanations/
26. Practical guide to SHAP analysis: Explaining supervised machine learning model predictions in drug development - PMC, accessed August 17, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11513550/
27. Calculating XGBoost Feature Importance | by Emily K Marsh - Medium, accessed August 17, 2025, https://medium.com/@emilykmarsh/xgboost-feature-importance-233ee27c33a4
28. Model Interpretability - XGBoost + SHAP - Kaggle, accessed August 17, 2025, https://www.kaggle.com/code/bennyfung/model-interpretability-xgboost-shap
29. Evaluating the CIC IDS-2017 Dataset Using Machine Learning Methods and Creating Multiple Predictive Models in the Statistical Computing Language R - IRJAES, accessed August 17, 2025, http://irjaes.com/wp-content/uploads/2020/10/IRJAES-V5N2P184Y20.pdf
30. A Comparative Analysis of Machine Learning and Deep Learning Approaches for Network Intrusion Detection - University of Idaho, accessed August 17, 2025, https://verso.uidaho.edu/view/pdfCoverPage?instCode=01ALLIANCE_UID&filePid=13338758540001851&download=true
31. Feature Selection using XGBoost - Medium, accessed August 17, 2025, https://medium.com/@dhanyahari07/feature-selection-using-xgboost-f0622fb70c4d
32. Use XGBoost Feature Importance for Feature Selection, accessed August 17, 2025, https://xgboosting.com/use-xgboost-feature-importance-for-feature-selection/
33. Features in cic ids 2017 dataset | Download Scientific Diagram - ResearchGate, accessed August 17, 2025, https://www.researchgate.net/figure/Features-in-cic-ids-2017-dataset_tbl1_343850781
34. (PDF) The Improved Network Intrusion Detection Techniques Using the Feature Engineering Approach with Boosting Classifiers - ResearchGate, accessed August 17, 2025, https://www.researchgate.net/publication/386896418_The_Improved_Network_Intrusion_Detection_Techniques_Using_the_Feature_Engineering_Approach_with_Boosting_Classifiers
35. How to Interpret Your XGBoost Model: A Practical Guide to Feature Importance - MachineLearningMastery.com, accessed August 17, 2025, https://machinelearningmastery.com/how-to-interpret-your-xgboost-model-a-practical-guide-to-feature-importance/

36. From Code to Clarity: XGBoost with SHAP Explained | DigitalOcean, accessed August 17, 2025, https://www.digitalocean.com/community/tutorials/xgboost-guide-shap-analysis-code-demo
37. Network Intrusion dataset(CIC-IDS- 2017) - Kaggle, accessed August 17, 2025, https://www.kaggle.com/datasets/chethuhn/network-intrusion-dataset
38. xgb.importance function - RDocumentation, accessed August 17, 2025, https://www.rdocumentation.org/packages/xgboost/versions/0.6-4/topics/xgb.importance
39. Basic SHAP Interaction Value Example in XGBoost, accessed August 17, 2025, https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/tree_based_models/Basic%20SHAP%20Interaction%20Value%20Example%20in%20XGBoost.html
40. Census income classification with XGBoost — SHAP latest documentation - Read the Docs, accessed August 17, 2025, https://shap.readthedocs.io/en/latest/example_notebooks/tabular_examples/tree_based_models/Census%20income%20classification%20with%20XGBoost.html
41. A Study on NSL-KDD Dataset for Intrusion Detection System Based ..., accessed August 17, 2025, https://e-tarjome.com/storage/btn_uploaded/2019-07-13/1563006133_9702-etarjome-English.pdf
42. NSL-KDD dataset features | Download Table - ResearchGate, accessed August 17, 2025, https://www.researchgate.net/figure/NSL-KDD-dataset-features_tbl1_302594395
43. Analysis of NSL-KDD for the Implementation of Machine Learning in Network Intrusion Detection System - Journal, accessed August 17, 2025, https://journal.ittelkom-pwt.ac.id/index.php/inista/article/download/1389/432/
44. NSL-KDD Anomaly detection - Kaggle, accessed August 17, 2025, https://www.kaggle.com/code/avk256/nsl-kdd-anomaly-detection
45. A Deeper Dive into the NSL-KDD Data Set | by Gerry Saporito | TDS Archive - Medium, accessed August 17, 2025, https://medium.com/towards-data-science/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657
46. Intrusion detection evaluation dataset (CIC-IDS2017) - University of New Brunswick, accessed August 17, 2025, https://www.unb.ca/cic/datasets/ids-2017.html
47. UNSW-NB15 - Kaggle, accessed August 17, 2025, https://www.kaggle.com/datasets/alextamboli/unsw-nb15
48. UNSW_NB15 - Kaggle, accessed August 17, 2025, https://www.kaggle.com/datasets/mrwellsdavid/unsw-nb15
49. The UNSW-NB15 dataset - Research Data Australia, accessed August 17, 2025, https://researchdata.edu.au/the-unsw-nb15-dataset/1957529
50. CIC UNSW-NB15 Augmented Dataset - University of New Brunswick, accessed August 17, 2025, https://www.unb.ca/cic/datasets/cic-unsw-nb15.html
51. Troubleshooting an Intrusion Detection Dataset: the CICIDS2017 Case Study -

DistriNet Research Unit, accessed August 17, 2025, https://intrusion-detection.distrinet-research.be/WTMC2021/Resources/wtmc2021_Engelen_Troubleshooting.pdf
52. UNSW-NB15 Computer Security Dataset: Analysis through Visualization - arXiv, accessed August 17, 2025, https://arxiv.org/pdf/2101.05067
53. Intrusion-Detection-using-Machine-Learning-on-NSL--KDD-dataset/IDS.ipynb at master, accessed August 17, 2025, https://github.com/Deepthi10/Intrusion-Detection-using-Machine-Learning-on-NSL--KDD-dataset/blob/master/IDS.ipynb
54. EDA of NSL-KDD Intrusion Detection Network Anomaly - Kaggle, accessed August 17, 2025, https://www.kaggle.com/code/andira/eda-of-nsl-kdd-intrusion-detection-network-anomaly
55. The Improved Network Intrusion Detection Techniques Using the Feature Engineering Approach with Boosting Classifiers - MDPI, accessed August 17, 2025, https://www.mdpi.com/2227-7390/12/24/3909
56. The explanation names of features in CICIDS2017 Dataset ..., accessed August 17, 2025, https://www.researchgate.net/figure/The-explanation-names-of-features-in-CICIDS2017-Dataset_tbl2_345237524
57. Feature Engineering in Machine Learning-Based Intrusion Detection Systems for OT Networks - IEEE Computer Society, accessed August 17, 2025, https://www.computer.org/csdl/proceedings-article/smartcomp/2023/228100a361/1PpUeLYlfxe
58. Mastering K-Nearest Neighbors (KNN): A Comprehensive Guide with Detailed Code | by Ebad Sayed | Medium, accessed August 17, 2025, https://medium.com/@sayedebad.777/mastering-k-nearest-neighbors-knn-aa4b2ffca68b
59. A Gentle Introduction to Imbalanced Classification ..., accessed August 17, 2025, https://machinelearningmastery.com/what-is-imbalanced-classification/
60. Feature Importance-Based Backdoor Attack in NSL-KDD - MDPI, accessed August 17, 2025, https://www.mdpi.com/2079-9292/12/24/4953
61. Advanced Evaluation Metrics for Imbalanced Classification Models | by Rajneesh Tiwari | CueNex | Medium, accessed August 17, 2025, https://medium.com/cuenex/advanced-evaluation-metrics-for-imbalanced-classification-models-ee6f248c90ca
62. Performance Metrics Deep Dive - Ultralytics YOLO Docs, accessed August 17, 2025, https://docs.ultralytics.com/guides/yolo-performance-metrics/
63. Imbalanced class distribution and performance evaluation metrics: A systematic review of prediction accuracy for determining model performance in healthcare systems - PubMed Central, accessed August 17, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10688675/
64. PERFORMANCE EVALUATION OF MACHINE LEARNING ... - arXiv, accessed August 17, 2025, https://arxiv.org/pdf/2310.00594
65. Best techniques and metrics for Imbalanced Dataset - Kaggle, accessed August

17, 2025, https://www.kaggle.com/code/marcinrutecki/best-techniques-and-metrics-for-imbalanced-dataset
66. Accuracy, precision, and recall in multi-class classification - Evidently AI, accessed August 17, 2025, https://www.evidentlyai.com/classification-metrics/multi-class-metrics
67. Classification/evaluation metrics for highly imbalanced data - Stats Stackexchange, accessed August 17, 2025, https://stats.stackexchange.com/questions/222558/classification-evaluation-metrics-for-highly-imbalanced-data
68. Classification on imbalanced data | TensorFlow Core, accessed August 17, 2025, https://www.tensorflow.org/tutorials/structured_data/imbalanced_data
69. The dataset comparison of KDD99, UNSW-NB15, CICIDS-2017, and HIKARI-2021 [68]., accessed August 17, 2025, https://www.researchgate.net/figure/The-dataset-comparison-of-KDD99-UNSW-NB15-CICIDS-2017-and-HIKARI-2021-68_tbl3_354172973
70. Explaining XGBoost model predictions with SHAP values - GitHub Pages, accessed August 17, 2025, https://samuel-book.github.io/samuel-2/samuel_shap_paper_1/xgb_with_feature_selection/03_xgb_combined_shap_key_features.html
71. 1.6. Nearest Neighbors — scikit-learn 1.7.1 documentation, accessed August 17, 2025, https://scikit-learn.org/stable/modules/neighbors.html