Explainable Brute-Force Detection Model

Ruizhe Jiang, Tom Shangguan, Karankumar Mageswaran

Abstract

Modern deep learning models achieve high accuracy in network security but suffer from a ”black box” problem that limits their practical adoption - security professionals can’t understand how these systems make decisions. We address this by building a pipeline that explains network traffic using human-understandable cybersecurity concepts. Our system converts raw PCAP files into natural language descriptions using a Large Language Model, then maps these summaries to 45 curated cybersecurity concepts derived from academic literature via LLM analysis and Gemini embeddings. We train a concept-mapping model on top of a fine tuned netFound backbone to learn this mapping directly from traffic data. Our evaluation demonstrates that the model consistently identifies stable and relevant core concepts for brute-force attacks, validating our approach and representing a significant step toward trustworthy, explainable network security systems.

1 Introduction

The widespread use of deep learning has created highly effective Network Intrusion Detection Systems (NIDS). However, these models’ superior performance is often limited by a major problem: they’re difficult to interpret, debug, and trust. Traditional explainability methods focus on identifying important low-level features (like specific header values), but these often fail to provide meaningful insights because they don’t match how human security experts think about complex, multi-packet threats. This gap between what the model sees and what humans understand remains a major barrier to using advanced LLM in critical security operations.

To bridge this gap, we use concept-based explanations, an approach that suggests a model becomes trustworthy when its decisions can be explained using high-level, human-understandable concepts. Instead of reasoning about individual bytes or packet fields, our approach focuses on abstract concepts like ”High Frequency of Failed Login Attempts” or ”Packet Burst Patterns.” Our main contribution is designing, building, and testing a complete, end-to-end pipeline that puts this idea into practice, specifically for brute-force attack detection.

By creating a system that explains network events in terms that make sense to human operators, we take a real step towards building more transparent, reliable, and trustworthy security infrastructure.

2 Background and Motivation

Deep learning has significantly improved NIDS by enabling them to detect complex cyber threats like brute-force attacks more effectively than traditional signature-based methods. These advanced models can identify subtle patterns in network traffic with greater accuracy and adaptability. However, the ”black box” nature of deep learning creates a major problem: security professionals often can’t understand why the system made certain decisions, making it hard to trust and act on the results. Current explainability methods like feature importance metrics and saliency maps focus on low-level data features, but they don’t provide insights that match how human analysts actually think about security threats. This gap makes it difficult for deep learning models and cybersecurity experts to work together effectively, especially when dealing with sophisticated multi-stage attacks. To solve this problem, researchers are developing concept-based explainability approaches that translate model decisions into human-understandable concepts like ”High Frequency of Failed Login Attempts” or ”Unusual Access Times.” Our goal is to create NIDS that are both highly accurate at detecting threats and able to explain their reasoning in ways that security professionals can easily understand and use.

3 Methodology

We implemented a fully automated, end-to-end pipeline: raw HTTP PCAPs are first segmented into direction-based bursts, core protocol headers and timing metadata are extracted, and everything is converted into boundary-marked token sequences—no manual feature engineering required. These token+metadata sequences are then fed into the pretrained netFound Transformer [1] (a hierarchical burst encoder followed by a flow encoder) and the entire model is fine-tuned end-to-end using AdamW with learning-rate warm-up/decay, mixed-precision training, and early stopping on a validation set. On the held-out test set, this yields 98.85% accuracy and 98.91% macro-F1, demonstrating the approach’s efficiency and robustness for HTTP brute-force detection.

The second part implements a multi-stage pipeline to train a mapper function that maps raw network packets to human-defined concepts [5]. We began by creating a set of security concepts and natural-language descriptions for a series of attack and benign packets. Using the Gemini embedding model, we calculated the similarity between every concept and every description, converting this human knowledge into a set of numerical, binned ”concept scores” which became our ground-truth labels. The core of our methodology was then to train a mapping function that, using features from a fine-tuned netFound, learns to predict these concept scores directly from network traffic. The result is a model that can not only classify a potential threat but also explain its reasoning by identifying which specific, human-understandable concepts are present in the network activity.

4 Implementation

4.1 Brute-force Detection Model Finetuning

We fine-tune the pre-trained `netFound-640M-base` model on Professor Arpit Gupta’s HTTP login PCAP corpus. Below we summarize the model architecture, data, preprocessing pipeline, and fine-tuning setup.

4.1.1 Model Architecture

The `netFound-640M-base` checkpoint consists of two hierarchical Transformer stages:

- **Burst Encoder (12 layers).** Each burst (up to 20 packets) is tokenized into ≈730 tokens (including a leading `[CLS burst]` marker and boundary `[SEP]` tokens). The burst encoder applies self-attention over those tokens and outputs a 768-dim embedding for the `[CLS·burst]` token, summarizing all packet-level information (EtherType, IPv4/TCP header fields, first bytes of HTTP payload, direction, inter-arrival time).
- **Flow Encoder (12 layers).** The five 768-dim burst embeddings are concatenated with a final `[CLS·flow]` token (six total tokens). The flow encoder self-attends over these six tokens to produce a 768-dim flow representation (the `[CLS·flow]` output).

All token embeddings (≈ 12,000 distinct 2-byte IDs) and four metadata embeddings (burst ID, packet direction, inter-arrival time, absolute position) are summed before each Transformer layer. A two-layer MLP classification head (hidden size = 512, GELU → softmax) maps the final `[CLS flow]` output to a binary prediction.

4.1.2 Dataset

Our dataset comprises approximately 393,424 single-flow PCAPs collected by Gupta *et al.*:

- **Malicious Flows.** Each PCAP captures a Patator-generated HTTP POST with incorrect credentials, followed by one or more HTTP 4xx responses.
- **Benign Flows.** Each PCAP contains one successful HTTP login (HTTP 200 OK), captured via an automated browser script targeting the same endpoint.

We split at the PCAP (flow) level into:

- **Train Set (70%)** Stratified to preserve the ∼64%/36% benign/malicious ratio.
- **Validation Set (10% of Train)** Randomly held out from the training portion, maintaining stratification.
- **Test Set (30%)** Held completely to measure the final performance.

Because each PCAP represents a fresh TCP handshake, there is no overlap in five-tuple identifiers across splits.

4.1.3 Preprocessing Pipeline

Each PCAP is converted to a fixed-length token+metadata sequence on-the-fly as follows:

1. **Burst Segmentation (exactly 5 bursts).** Read the single TCP flow; group packets into bursts by direction (client→server vs. server→client) and a 10 ms inter-packet gap. If fewer than 5 bursts exist, pad with empty bursts; if more (rare retransmissions), merge extras so exactly 5 bursts remain.
2. **Packet-Level Feature Extraction.** For each packet, extract:
    - Ethernet EtherType (2 bytes), IPv4 header (IHL, total length, DSCP/ECN, flags, TTL), TCP header (src/dst ports, truncated sequence/ack numbers, flags, window, up to 8 bytes of TCP options), and first 12 bytes of HTTP payload.
    - Two metadata values: packet direction (0/1) and inter-arrival time in µs (capped at 65,535).
3. **Burst-Level Flattening (1 040 bytes).** Concatenate each burst’s 20×≈52B packet vectors into a 1040B block. Prepend a `[CLS·burst]` token (ID = 1) and insert `[SEP]` tokens (ID = 2) in fixed byte offsets that mark IP - TCP and TCP - HTTP boundaries.
4. **2-Byte Tokenization (≈ 2 600 tokens per flow).** Build a vocabulary of all 2-byte sequences appearing across the dataset (≈ 12000 unique IDs). Split each 5200B flow block (5 bursts × 1040B) into 2600 contiguous 2-byte chunks; map each chunk to its 16-bit token ID (unseen pairs → `[UNK]` ID = 65535). This yields ≈ 2600 tokens, plus 5 `[CLS burst]` and a final `[CLS flow]` (ID = 3), for a total of exactly **3 630 tokens** per flow.
5. **Metadata Arrays (length 3 630).** In parallel, we construct four arrays of length 3 630:

(a) *Burst ID* (1–5 for tokens in bursts; 0 for `[CLS flow]`),
(b) *Direction* (0/1 for packet tokens; –1 for `[CLS]` and `[SEP]`),
(c) *Inter-arrival Time* (0–65535 for packet tokens; 0 for markers),
(d) *Position* (0–3629).

These are embedded (size 768) and summed with the token embeddings before each Transformer layer.

4.1.4 Fine-Tuning Configuration

We update all 24 Transformer layers and attach a two-layer MLP head (hidden = 512, GELU $\to$ softmax). Key hyperparameters:

- **Batch Size:** 28 flows/GPU $\times$ 4 GPUs = 112 total.
- **Optimizer:** AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$), weight decay = 0.01.
- **Learning Rate:** 2×10^{-5}, with linear warm-up over the first 10 % of total steps, then linear decay to 0.
- **Epochs:** Up to 7; early stopping if validation macro-F_1 does not improve for 2 consecutive epochs.
- **Gradient Clipping:** $\|g\|_2 \leq 1.0$.
- **Mixed Precision:** FP16 with dynamic loss scaling.
- **Validation Split:** 10 % of train (27 540 flows), stratified by class.

After roughly 5 epochs, the best validation checkpoint achieves **98.91 %** accuracy (98.85 % macro-F_1) on the held-out test set.

4.2 Concept Mapping Pipeline

We followed the blueprint for concept-based explanations to create an interpretable representation of how the netFound classifies attack or benign network packages. Our architecture is instantiated as a concept mapping pipeline, composed of four primary stages: (4.2.1) Concept Generation, (4.2.2) Packet Level Description, (4.2.3) Description-Concept Embedding, and (4.2.4) Concept Mapping.

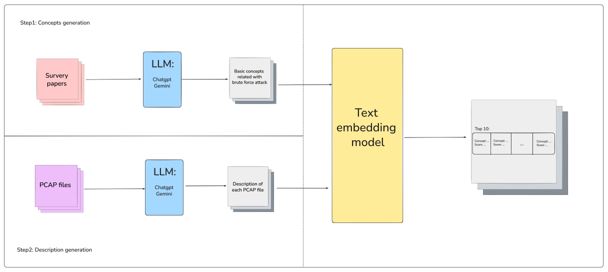

Figure 1: *Overview of concept- and description-generation.*

4.2.1 Stage 1: Concept Generation

To generate high-quality concepts, we first prompted a LLM to synthesize survey papers we chose on brute-force attacks and extract a candidate list of core concepts involved in the paper [2–4]. We then manually viewed this list to remove irrelevant or redundant concepts. This approach ensures the concepts generated are both representative and concise, forming a robust semantic basis for our mapping.

4.2.2 Stage 2: Packet Level Description

This stage is responsible for abstracting raw packet data into a human-readable description. The process involves two steps. First, we extract key header fields from each network packet, including timestamps, source/destination IPs, protocols, etc. The data payload of packets is intentionally ignored to improve processing speed. Second, the extracted header fields from all packets within a single capture file are provided as context to an LLM. The LLM is then prompted to generate a comprehensive summary describing the aggregate activity in that file. Overall, this step translates low-level, machine-readable data into a high-level, descriptive text suitable for semantic analysis.

4.2.3 Stage 3: Description-Concept Embedding

This stage quantifies the relationship between the traffic description and our predefined concepts. We use a *Gemini* embedding model ε to create vectors for our 45 concepts C (extracted in Stage 1) and for each PCAP file description x. We then calculate the cosine similarity between the description and each concept c and bin the result using a quantization function ψ_k. The process is represented as

$$Sc_c = \psi_k\left(\frac{\varepsilon(x)\cdot\varepsilon(c)}{\|\varepsilon(x)\|\,\|\varepsilon(c)\|}\right), \quad \forall c \in C. \tag{1}$$

Here, the quantization function ψ_k discretizes the continuous cosine-similarity score into one of k predefined classes or "bins," where k is a hyper-parameter. This step explores the relation between the description and the concepts. It yields a target concept-score vector for each description, providing the labeled data for the next stage.

4.2.4 Stage 4: Concept Mapping

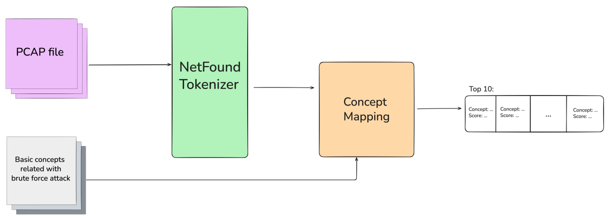

Figure 2: *Tokenization and concept-mapping at inference time.*

With the ground-truth concept scores (Sc_c) set, the final stage is to train a mapping function to directly predict these scores from internal embeddings of network packets. This enables the system to map low-level network states into high-level, human-interpretable concepts. To perform this task, we use the fine-tuned netFound model from 4.1 as a feature extractor. For any network packet x, the model produces an embedding $h(x)$, capturing the underlying characteristics. On top of this representation, we train the mapping function δ_θ, where θ are the learnable parameters of the mapping. This function outputs predictions over the k quantized bins for each concept. The mapping model consists of 45 independent heads, one for each concept. Each head is implemented as a classifier that outputs a categorical distribution over k bins. These heads are trained jointly using a multi-label classification objective, summing the per-concept Cross-Entropy losses. Formally, for an input x and its corresponding concept labels Sc, the loss is given by:

$$l(x,Sc) = \sum_{c}^{C} \left[-\log \left(\frac{e^{(\delta_\theta(h(x)),Sc_c)}}{\sum_{i=1}^{k} e^{(\delta_\theta(h(x)),i)}} \right) \right]$$

This expression computes the softmax probability assigned to the correct bin Sc for each concept c, and penalizes the model based on the log-loss. The total loss is minimized using mini-batch stochastic gradient descent, updating θ, while keeping the base embedding $h(x)$ fixed.

The output of δ_θ is a $C \times k$ matrix, where each row corresponds to the softmax distribution over bins for one concept. By training this mapping, the model learns to align internal embeddings with meaningful, interpretable concepts, thereby enabling concept-level explanation.

5 Results

To evaluate our trained Concept Mapping model, we performed inference on 100 distinct network packet captures known to contain brute-force attack activity. The objective was to assess whether the model could consistently identify a relevant and stable set of high-level concepts that accurately describe the nature of the traffic. The model's predictions, showing the top 10 concepts identified for each of the 100 different attack packets, demonstrate a remarkable consistency. As we found that for many packets, the model identified some common concepts, which may imply that these concepts are generally used to predict whether a network packet is benign or not.

This consistency suggests that our pipeline has successfully learned a stable and semantically relevant "concept fingerprint" for this specific type of attack. The most prominent concepts identified, such as "High Frequency of Failed Login Attempts" and "TCP Flag Patterns in Failed SSH Connections," were consistently assigned the highest similarity score, indicating a strong match. Supporting contextual concepts, like "Brute Force Activity on Non-Standard SSH Ports" and "Packet Burst Patterns in Attack Windows," were also consistently identified with a high, although slightly lower, score. This result validates that our model can reliably abstract low-level packet data into a meaningful and human-interpretable representation, providing a consistent high-level narrative for similar types of malicious network activity.

6 Conclusion

In this work, we built and tested an end-to-end pipeline that applies concept-based explanations to network security. We converted raw packet data to a set of human-understandable concepts, showing a potential path toward more interpretable systems compared to traditional feature-based models. Our results suggest that the trained concept-mapping model can identify a reasonably stable set of relevant concepts for brute-force attacks, indicating that a "concept fingerprint" might be learnable directly from network traffic. This research explores the potential of combining foundation models like LLMs and embedding models with domain-specific knowledge to improve model trustworthiness. While our current implementation focuses on mapping traffic to concepts, it provides a foundation for future work, including training a classifier on top of predicted concept scores to build a fully interpretable NIDS, expanding the concept set to cover more attack types, and testing the system's robustness against more sophisticated attacks. This work represents a step toward helping security analysts not only trust AI system predictions but also better understand how those predictions were made.

References

[1] GUTHULA, S., BELTIUKOV, R., BATTULA, N., GUO, W., GUPTA, A., AND MONGA, I. netfound: Foundation model for network security, 2025. (Cited on page 1.)

[2] JACOB, S., AND KUMAR, B. S. A survey on brute force attack on open functionality secured. *IOSR Journal of Computer Engineering (IOSR-JCE) 20*, 5 (September 2018), 01–06. Version IV. (Cited on page 3.)

[3] NISHA, H. S., RISHMA, M. A., MAGADUM, P., RAKSHITHA, S., AND KARTHIGA. Survey: Detection of brute force attacks on iot networks. *International Research Journal of Modernization in Engineering Technology and Science 6*, 3 (March 2024). (Cited on page 3.)

[4] OWENS, J., AND MATTHEWS, J. A study of passwords and methods used in brute-force ssh attacks. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (2008), p. 8. (Cited on page 3.)

[5] PATEL, S., HAN, D., NARODYSTKA, N., AND JYOTHI, S. A. Toward trustworthy learning-enabled systems with concept-based explanations. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks* (2024), pp. 60–67. (Cited on page 2.)