Proses Preprocessing Dataset ISCX 2016 VPN/Non-VPN:
Secara garis besar seperti berikut:

Dataset Preprocessing for Encrypted Network Traffic Analysis

The preprocessing of the ISCX 2016 VPN/Non-VPN dataset is a critical step for preparing the data for the multi-task learning model. This process involves converting raw packet-level data into a structured flow-level format and cleaning it to ensure high-quality input for the model. The methodology leverages a two-stage cleaning process initially proposed in a separate study and adds a final filtering step.

Initial Data Conversion and Cleaning

The initial steps of the preprocessing are adopted from the paper "Preprocessing and Analysis of an Open Dataset in Application Traffic Classification". This foundational process is crucial for refining the raw dataset.

1. **Packet to Flow Conversion**: The process begins by converting the raw `.pcap` files, which contain packet-level data, into bidirectional flows or sessions. This is accomplished using a tool called SplitCap, which groups packets into flows based on common session identifiers.
2. **Protocol-Based Flow Removal**: The first cleaning criterion involves analyzing the application-layer protocols within each flow. Flows containing protocols that are likely generated by the operating system or network configuration, rather than the application itself (such as NBSS, LLMNR, and DNS), are removed to reduce noise.
3. **TCP 3-Way Handshake Verification**: The second cleaning criterion focuses on the integrity of TCP flows. TCP flows that do not have a complete 3-way handshake (SYN, SYN-ACK, ACK) at the beginning of the session are discarded. This ensures that the model is trained on complete and valid TCP sessions, which is representative of real-world traffic.

After these two steps, the original 309,000 flows in the dataset are significantly reduced to 29,195 flows.

Additional Flow Filtering

Following the initial cleaning, a further filtering step is applied to remove what are considered non-essential flows for the research objectives.

- **Beacon Flow Removal**: The dataset is scanned for specific UDP flows that have a destination IP address of 255.255.255.255 and contain the string "Beacon~" in their payload. These flows are considered unnecessary for the traffic classification tasks and are removed.

After this final filtering, the dataset is further reduced to a total of **8,763 flows**.

Step 1: Obtain the Dataset and Necessary Tools

First, you need to get the raw data and the tools mentioned in the research.

1. **Download the Dataset**: Get the **ISCX VPN-nonVPN 2016 dataset**. This is the raw data in `.pcap` format.
2. **Get the Splitting Tool**: You'll need the **SplitCap** tool. This is used to break down the large `.pcap` files into individual files, one for each bidirectional flow.

Step 2: Split PCAPs into Bidirectional Flows

Before you can filter, you must convert the packet-level data into flow-level data.

- Use the **SplitCap** tool on the raw `.pcap` files from the dataset. This will generate a large number of smaller files, where each file represents a single bidirectional flow (or session). The original dataset has approximately 309,000 flows in total.

Step 3: Apply Automated Filtering Rules

This is the core of the cleaning process, where you'll run scripts to apply a series of rules to discard irrelevant flows.

Rule A: Filter by Application Protocol

The first script should remove flows that are not generated by the target applications but rather by the operating system or network services.

- **Action**: Automatically remove flows that use irrelevant protocols like **DNS, NBSS, and LLMNR**.

Rule B: Verify TCP 3-Way Handshake

Next, filter out incomplete or improperly captured TCP connections.

- **Action**: Automatically delete any TCP flow that has **more than four packets** but where the **first packet does not have the SYN flag set**. This ensures that you're only keeping flows from properly established TCP sessions. After applying rules A and B, you should have around 29,195 flows.

Rule C: Remove Specific UDP Broadcasts

This is the final filtering step, mentioned in the "Fast and Accurate Multi-Task Learning" paper, which gets you to the target flow count.

- **Action**: Perform a final analysis to remove specific, non-essential UDP flows. The script should delete flows that meet all of the following criteria:

- The protocol is **UDP**.
- The destination IP address is the broadcast address **255.255.255.255**.
- The payload contains the string **"Beacon~"**.

Reasoning mengapa mengubah raw .pcap dari dataset menjadi Bidirectional Flows or sessions:

What are .pcap files and flows?

First, let's clarify the terms:

- **`.pcap` file**: This is a raw data file that captures and stores individual network packets traveling over a network. Think of it as a detailed log of every single piece of data that passes through a certain point.
- **Bidirectional Flow (or Session)**: A flow represents a complete conversation between two endpoints on a network. It groups all packets that share the same key identifiers:
  - Source IP address
  - Destination IP address
  - Source port number
  - Destination port number
  - Protocol (like TCP or UDP)

By converting packets into flows, you shift from analyzing individual data fragments to looking at the entire conversation, which is much more informative.

Why is this conversion necessary?

1. **Context is Key** 🔑 Analyzing individual packets is like trying to understand a story by reading one word at a time. You miss the context and the bigger picture. By grouping packets into flows, you can see the entire interaction between two devices. This is especially important for understanding the behavior of applications, as they communicate through a series of back-and-forth messages.
2. **More Meaningful Features** Flow-based analysis allows you to extract more meaningful features that are not available at the packet level. These features can include:
   - **Flow duration**: How long the conversation lasted.
   - **Total bytes transferred**: The total amount of data exchanged.
   - **Packet inter-arrival times**: The time intervals between packets in the flow.
3. These statistical features are crucial for many network analysis tasks, including traffic classification and security monitoring.
4. **Efficiency and Scalability** 🚀 Processing every single packet on a busy network can be incredibly resource-intensive. Converting packets into flows significantly reduces the amount of data you need to analyze, as you're looking at aggregated information rather than every individual packet. This makes the analysis process faster and more scalable, which is essential for real-time applications.
5. **Handling Encrypted Traffic** With the rise of encryption, inspecting the payload

(the actual content) of packets is often impossible. Flow-based analysis provides a way to classify encrypted traffic by looking at the metadata and statistical properties of the flow, such as packet sizes and timing, without needing to decrypt the content.

In short, converting `.pcap` files to bidirectional flows allows for a more efficient, scalable, and context-aware analysis of network traffic, which is fundamental for tasks like application classification, network security, and performance monitoring.

Reasoning mengapa menggunakan SplitCap daripada (misalnya) membuat script python dengan Scapy untuk Splitting .pcap awal menjadi flows:

Feature	SplitCap (The Specialist)	Scapy Script (The DIY Approach)
Performance	Highly optimized (often written in C/C++) to process huge .pcap files quickly with low memory usage.	Can be very slow and memory-intensive, as Python has to manage a complex dictionary of all active flows in memory.
State Management	Handles session tracking, timeouts, and bidirectionality automatically and correctly.	You would need to manually implement all the logic to track sessions, handle timeouts, and correctly group A -> B and B -> A packets into one flow. This is prone to errors.
Simplicity	It's a simple, single-purpose command-line tool. You run one command, and it does the job perfectly.	You would need to write and debug a potentially complex script, reinventing a wheel that already exists in a more robust form.
Reproducibility	The research papers explicitly used SplitCap. Using the same tool is the best way to ensure you are accurately reproducing their methodology.	Your custom script might group or time-out flows differently than the researchers' tool, leading to a different number of flows and affecting your final results.

Praktiknya:
Dengan Ubuntu LTS 22.04 melakukan splitting raw .pcap awal menjadi flows/sessions:

Command-nya (untuk satu file .pcap):

```
mono ./SplitCap.exe -r NonVPN-PCAPs-01/facebook_audio4.pcap -o
split_flows/facebook_audio4 -s session -p 500
```

Struktur folder dataset:

```
NonVPN-PCAPs-01:
aim_chat_3a.pcap
aim_chat_3b.pcap
AlMchat1.pcap
AIMchat2.pcap
email1a.pcap
email1b.pcap
email2a.pcap
email2b.pcap
facebook_audio1a.pcap
facebook_audio1b.pcap
facebook_audio2a.pcap
facebook_audio2b.pcap
facebook_audio3.pcap
facebook_audio4.pcap
facebook_chat_4a.pcap
facebook_chat_4b.pcap
facebook_video1a.pcap
facebook_video1b.pcap
facebook_video2a.pcap
facebook_video2b.pcap
facebookchat1.pcap
facebookchat2.pcap
facebookchat3.pcap

NonVPN-PCAPs-02:
ftps_down_1a.pcap
ftps_down_1b.pcap
ftps_up_2a.pcap
ftps_up_2b.pcap
gmailchat1.pcap
gmailchat2.pcap
gmailchat3.pcap
hangout_chat_4b.pcap
hangouts_audio1a.pcap
hangouts_audio 1b.pcap
hangouts_audio2a.pcap
hangouts_audio2b.pcap
hangouts_audio3.pcap
hangouts_audio4.pcap
hangouts_chat_4a.pcap
hangouts_video 1b.pcap
hangouts_video2a.pcap
hangouts_video2b.pcap
icq_chat_3a.pcap
icq_chat_3b.pcap
ICQchat1.pcap
ICQchat2.pcap
netflix1.pcap
```

```
netflix2.pcap
netflix3.pcap
netflix4.pcap

NonVPN-PCAPs-03:
scp1.pcap
scpDown1.pcap
scpDown2.pcap
scpDown3.pcap
scpDown4.pcap
scpDown5.pcap
scpDown6.pcap
scpUp1.pcap
scpUp2.pcap
scpUp3.pcap
scpUp5.pcap
scpUp6.pcap
sftp_down_3a.pcap
sftp_down_3b.pcap
sftp_up_2a.pcap
sftp_up_2b.pcap
sftp1.pcap
sftpDown1.pcap
sftpDown2.pcap
sftpUp1.pcap
skype_audio1a.pcap
skype_audio1b.pcap
skype_audio2a.pcap
skype_audio2b.pcap
skype_audio3.pcap
skype_audio4.pcap
skype_chat1a.pcap
skype_chat1b.pcap
skype_file1.pcap
skype_file2.pcap
skype_file3.pcap
skype_file4.pcap
skype_file5.pcap
skype_file6.pcap
skype_file7.pcap
skype_file8.pcap
skype_video1a.pcap
skype_video1b.pcap
skype_video2a.pcap
skype_video2b.pcap
spotify1.pcap
spotify2.pcap
spotify3.pcap
spotify4.pcap
vimeo1.pcap
vimeo2.pcap
vimeo3.pcap
vimeo4.pcap
voipbuster_4a.pcap
voipbuster_4b.pcap
```

```
voipbuster1b.pcap
voipbuster2b.pcap
voipbuster3b.pcap
youtube1.pcap
youtube2.pcap
youtube3.pcap
youtube4.pcap
youtube5.pcap
youtube6.pcap
youtubeHTML5_1.pcap

VPN-PCAPS-01:
vpn_aim_chat1a.pcap
vpn_aim_chat1b.pcap
vpn_bittorrent.pcap
vpn_email2a.pcap
vpn_email2b.pcap
vpn_facebook_audio2.pcap
vpn_facebook_chat1a.pcap
vpn_facebook_chat1b.pcap
vpn_ftps_A.pcap
vpn_ftps_B.pcap
vpn_hangouts_audio1.pcap
vpn_hangouts_audio2.pcap
vpn_hangouts_chat1a.pcap
vpn_hangouts_chat1b.pcap

VPN-PCAPs-02:
vpn_icq_chat1a.pcap
vpn_icq_chat1b.pcap
vpn_netflix_A.pcap
vpn_sftp_A.pcap
vpn_sftp_B.pcap
vpn_skype_audio1.pcap
vpn_skype_audio2.pcap
vpn_skype_chat1a.pcap
vpn_skype_chat1b.pcap
vpn_skype_files1a.pcap
vpn_skype_files1b.pcap
vpn_spotify_A.pcap
vpn_vimeo_A.pcap
vpn_vimeo_B.pcap
vpn_voipbuster1a.pcap
vpn_voipbuster1b.pcap
vpn_youtube_A.pcap
```

Melakukan automasi dengan script python untuk melakukan command di atas untuk semua file .pcap dari dataset:

```
import os
import subprocess

```

```
# --- Configuration ---

# The name of the executable for SplitCap
SPLITCAP_EXECUTABLE = './SplitCap.exe'

# List of your dataset directories
DATASET_DIRS = [
    'NonVPN-PCAPs-01',
    'NonVPN-PCAPs-02',
    'NonVPN-PCAPs-03',
    'VPN-PCAPS-01',
    'VPN-PCAPS-02'
]

# The main output directory where all split flows will be stored
MAIN_OUTPUT_DIR = 'split_flows'

# --- Script Logic ---

def run_splitcap(pcap_file_path, output_dir):
    """
    Constructs and runs the SplitCap command for a single pcap
file.
    Splits the file into bidirectional flows (sessions).

    *** UPDATED: Includes the -p argument to limit parallel file
handles. ***
    """
    # From the documentation, the argument for bidirectional
flows is 'session'
    split_argument = 'session'

    # Set a limit for parallel sessions to prevent "Too many open
files" error
    parallel_sessions_limit = '500'

    # The command to execute in WSL
    command = [
        'mono',
        SPLITCAP_EXECUTABLE,
        '-r', pcap_file_path,
        '-o', output_dir,
        '-s', split_argument,
        '-p', parallel_sessions_limit  # <-- THIS IS THE NEW
ARGUMENT
    ]

    print(f"-> Processing: {pcap_file_path}")
    print(f"   Command: {' '.join(command)}")

    try:
        # Execute the command
        result = subprocess.run(
            command,
            check=True,        # This will raise an exception if
```

```
the command fails
            capture_output=True,
            text=True
        )
        print(f"   Success! Output saved to: {output_dir}")
        # Uncomment the line below if you want to see the full
output from SplitCap
        # print(result.stdout)
    except FileNotFoundError:
        print("\n[ERROR] 'mono' command not found.")
        print("Please ensure Mono is installed and accessible in
your WSL environment.")
        print("Installation command: sudo apt install
mono-runtime")
        return False
    except subprocess.CalledProcessError as e:
        # This error is triggered if SplitCap returns a non-zero
exit code (an error)
        print(f"\n[ERROR] SplitCap failed for file:
{pcap_file_path}")
        print(f"   Return Code: {e.returncode}")
        print(f"   Error Output:\n{e.stderr}")
        return False
    return True

def main():
    """
    Main function to find all pcap files and process them.
    """
    print("--- Starting PCAP to Bidirectional Flow Splitting
Process ---")

    # Check if SplitCap.exe exists
    if not os.path.exists(SPLITCAP_EXECUTABLE):
        print(f"[FATAL ERROR] '{SPLITCAP_EXECUTABLE}' not found
in the current directory.")
        return

    # Create the main output directory if it doesn't exist
    os.makedirs(MAIN_OUTPUT_DIR, exist_ok=True)
    print(f"Main output directory is '{MAIN_OUTPUT_DIR}'\n")

    total_files = 0
    success_count = 0

    # Iterate through each dataset directory
    for dir_name in DATASET_DIRS:
        if not os.path.isdir(dir_name):
            print(f"[WARNING] Directory '{dir_name}' not found.
Skipping.")
            continue

        print(f"--- Scanning directory: {dir_name} ---")
        # Find all files ending with .pcap in the directory
        for filename in os.listdir(dir_name):
```

```
            if filename.endswith('.pcap'):
                total_files += 1
                pcap_path = os.path.join(dir_name, filename)

                # Create a specific output directory for this
pcap file's flows
                # Example: 'split_flows/aim_chat_3a'
                output_sub_dir_name =
os.path.splitext(filename)[0]
                output_path = os.path.join(MAIN_OUTPUT_DIR,
output_sub_dir_name)

                # Create the directory for the split files
                os.makedirs(output_path, exist_ok=True)

                # Run SplitCap on the file
                if run_splitcap(pcap_path, output_path):
                    success_count += 1
                print("-" * 20) # Separator

    print("\n--- Processing Complete ---")
    print(f"Successfully processed {success_count} out of
{total_files} .pcap files.")

if __name__ == '__main__':
    main()
```

Total flow menjadi:
Non-VPN:

291.721 Files, 109 Folders

VPN:

18.468 Files, 31 Folders

Setelah menjadi flows, kemudian dilakukan filtering kembali dengan kriteria yang dijelaskan di awal. Semua flow di-upload ke GDrive, lalu filtering dilakukan di Google Colab dengan script berikut:

```
import os
import shutil
import logging
# --- Scapy Imports ---
# Import only the most basic and stable layers
from scapy.all import rdpcap, TCP, UDP, IP
from scapy.layers.dns import DNS
from scapy.layers.dhcp import DHCP, BOOTP
from scapy.layers.ntp import NTP
```

```
# --- Configuration ---

# Directory containing the split pcap files from Step 2 (~309k flows)
INPUT_DIR = '/content/drive/MyDrive/1
Skripsi/Dataset/ISCX-VPN-NonVPN-2016/split_flows_final'

# Directory where the final cleaned pcap files will be saved (~8.7k
flows)
OUTPUT_DIR = '/content/drive/MyDrive/1
Skripsi/Dataset/ISCX-VPN-NonVPN-2016/cleaned_flows_final'

# --- Rule A: Protocol Filtering ---
# DISALLOWED_PROTOCOLS: Check for protocols that Scapy can easily
identify as layers.
DISALLOWED_PROTOCOLS = {
    DNS,
    NTP,
    DHCP,
    BOOTP
}
# DISALLOWED_UDP_PORTS: A more robust way to block protocols based on
their standard ports.
DISALLOWED_UDP_PORTS = {
    137,   # NBNS (NetBIOS Name Service)
    1900,  # SSDP
    5353,  # MDNS
    5355   # LLMNR
}

# --- Logging Setup ---
logging.basicConfig(
    filename='filtering_log_final.txt',
    level=logging.INFO,
    filemode='w',  # Overwrite the log file on each run
    format='%(asctime)s - %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S'
)


def is_flow_valid_final(pcap_file_path):
    """
    Analyzes a single flow pcap file and applies ALL filtering rules
```

```
(A, B, and C)
    to determine if it belongs in the final dataset.
    Returns a tuple: (is_valid, reason_for_invalidation)
    """
    try:
        packets = rdpcap(pcap_file_path)
    except Exception as e:
        return False, f"Scapy could not read file. Error: {e}"

    if not packets:
        return False, "File is empty or corrupt"

    # --- Implement ALL Filtering Rules ---
    first_packet = packets[0]

    # Rule B: TCP 3-Way Handshake Check (applies only to TCP flows)
    if TCP in first_packet:
        if len(packets) > 4 and 'S' not in first_packet[TCP].flags:
            return False, "Rule B Failed: Invalid TCP handshake
start"

    # Loop through all packets to check Rules A and C
    for pkt in packets:
        # Rule A (Part 1): Check for disallowed application-layer
protocols by Scapy layer
        for protocol_layer in DISALLOWED_PROTOCOLS:
            if protocol_layer in pkt:
                return False, f"Rule A Failed: Disallowed protocol
({protocol_layer.name})"

        if UDP in pkt:
            # Rule A (Part 2): Check for disallowed UDP protocols by
their well-known port numbers
            if pkt[UDP].sport in DISALLOWED_UDP_PORTS or
pkt[UDP].dport in DISALLOWED_UDP_PORTS:
                return False, f"Rule A Failed: Disallowed UDP port
used"

            # Rule C: UDP Beacon Check
            # Check for broadcast UDP packets containing "Beacon~" in
the payload
            if IP in pkt and pkt[IP].dst == '255.255.255.255':
```

```
                try:
                    payload = bytes(pkt[UDP].payload)
                    if b'Beacon~' in payload:
                        return False, "Rule C Failed: UDP Beacon flow
detected"
                except Exception:
                    # Payload might be empty or malformed, safe to
ignore and continue
                    pass

    # If the flow has passed all the checks, it's valid
    return True, "Valid"


def main():
    """
    Main function to iterate through all flow files and filter them
to the final dataset.
    """
    print("--- Starting Combined Filtering (to get ~8,763 flows)
---")

    if not os.path.isdir(INPUT_DIR):
        print(f"[FATAL ERROR] Input directory '{INPUT_DIR}' not
found.")
        print("Please make sure you have run Step 2 successfully.")
        return

    os.makedirs(OUTPUT_DIR, exist_ok=True)
    print(f"Final cleaned flows will be saved in: '{OUTPUT_DIR}'")
    print(f"A detailed log will be saved to:
'filtering_log_final.txt'")
    print("\nThis process will take a very long time. Please be
patient.\n")

    total_flows = 0
    kept_flows = 0
    discarded_flows = 0

    # Walk through the nested directory structure of split_flows
    for root, _, files in os.walk(INPUT_DIR):
        for filename in files:
```

```
            if filename.endswith('.pcap'):
                total_flows += 1
                pcap_path = os.path.join(root, filename)

                is_valid, reason = is_flow_valid_final(pcap_path)

                if is_valid:
                    kept_flows += 1
                    try:
                        shutil.copy(pcap_path, OUTPUT_DIR)
                    except Exception as e:
                        log_msg = f"ERROR COPYING: {pcap_path} |
Error: {e}"
                        print(log_msg)
                        logging.error(log_msg)
                else:
                    discarded_flows += 1
                    logging.info(f"DISCARDED: {pcap_path} | Reason:
{reason}")

                # Print a progress update every 1000 files processed
                if total_flows % 1000 == 0:
                    print(f"Processed: {total_flows} | Kept:
{kept_flows} | Discarded: {discarded_flows}")

    print("\n--- Final Filtering Complete ---")
    print(f"Total flows processed: {total_flows}")
    print(f"Total flows kept: {kept_flows}")
    print(f"Total flows discarded: {discarded_flows}")
    print(f"\nYour final, cleaned dataset is now available in the
'{OUTPUT_DIR}' directory.")
    print(f"The final count should be near the target of 8,763.")

if __name__ == '__main__':
    main()
```