In [2]:
```
from tqdm import tqdm
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Subset
from urllib.parse import urlparse
from three_datasets import all_dataset
import random
tqdm.pandas()
seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

print(f"✅ Seed fixed to {seed}")
```

	label	url
0	2	https://blog.sockpuppet.us/
1	2	https://blog.apiki.com/seguranca/
2	1	http://autoecole-lauriston.com/a/T0RVd056QXlNe...
3	1	http://chinpay.site/index.html?hgcFSE@E$Z*DFcG...
4	2	http://www.firstfivenebraska.org/blog/article/...

```
c:\Users\rrpra\Documents\Github\git_testing\email_and_url_classifier_using_nlp_feature_extraction_sem_5_project\three_datasets.py:
70: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the intermedi
ate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the o
riginal `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy

  df4['label'][df4['label'] == 1] = 0
c:\Users\rrpra\Documents\Github\git_testing\email_and_url_classifier_using_nlp_feature_extraction_sem_5_project\three_datasets.py:
71: FutureWarning: ChainedAssignmentError: behaviour will change in pandas 3.0!
You are setting values through chained assignment. Currently this works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to update the original DataFrame or Series, because the intermedi
ate object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the assignment in a single step and ensure this keeps updating the o
riginal `df`.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy

  df4['label'][df4['label'] == 2] = 1
c:\Users\rrpra\Documents\Github\git_testing\email_and_url_classifier_using_nlp_feature_extraction_sem_5_project\three_datasets.py:
85: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versu
s-a-copy
  df['label'] = df['label'].apply(lambda x: 1 if x in phishing_labels else 0)
```

✅ Seed fixed to 42

In [3]:
```python
from transformers import AutoTokenizer,  T5EncoderModel
import torch
import numpy as np

model_name = "google/canine-s"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = T5EncoderModel.from_pretrained(model_name)
model.eval()
def get_byt5_embedding(url):
    # Tokenize the input
    inputs = tokenizer(url, return_tensors="pt", truncation=True, padding=True, max_length=50)

    # Get encoder output (not generate)
    with torch.no_grad():
        outputs = model(**inputs)
        # outputs.last_hidden_state: [batch_size, seq_len, hidden_dim]

        # Mean pooling across sequence
        emb = outputs.last_hidden_state.mean(dim=1).reshape(-1)
    return emb.cpu().numpy()

# Example
#url = "http://paypal-login-secure-update.com"
#vector = get_byt5_embedding(url)
#print("Embedding shape:", vector.shape)  # e.g. (512,) or (768,) depending on model size
```

```
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/attr_va
lue.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/tensor.
proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/resourc
e_handle.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/tensor_
shape.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/types.p
roto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/full_ty
pe.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/functio
n.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/node_de
f.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/op_def.
proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/graph.p
roto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
```

```
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/graph_d
ebug_info.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/version
s.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/protobuf/config.p
roto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at xla/tsl/protobuf/coordination_con
fig.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/cost_gr
aph.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/step_st
ats.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/allocat
ion_description.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/framework/tensor_
description.proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/protobuf/cluster.
proto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\protobuf\runtime_version.py:98: UserWarning: Proto
buf gencode version 5.28.3 is exactly one major version older than the runtime version 6.31.1 at tensorflow/core/protobuf/debug.pr
oto. Please update the gencode to avoid compatibility violations in the next runtime release.
  warnings.warn(
You are using a model of type canine to instantiate a model of type t5. This is not supported for all configurations of models and
```

```
can yield errors.
Some weights of T5EncoderModel were not initialized from the model checkpoint at google/canine-s and are newly initialized: ['enco
der.block.0.layer.0.SelfAttention.k.weight', 'encoder.block.0.layer.0.SelfAttention.o.weight', 'encoder.block.0.layer.0.SelfAttent
ion.q.weight', 'encoder.block.0.layer.0.SelfAttention.relative_attention_bias.weight', 'encoder.block.0.layer.0.SelfAttention.v.we
ight', 'encoder.block.0.layer.0.layer_norm.weight', 'encoder.block.0.layer.1.DenseReluDense.wi.weight', 'encoder.block.0.layer.1.D
enseReluDense.wo.weight', 'encoder.block.0.layer.1.layer_norm.weight', 'encoder.block.1.layer.0.SelfAttention.k.weight', 'encoder.
block.1.layer.0.SelfAttention.o.weight', 'encoder.block.1.layer.0.SelfAttention.q.weight', 'encoder.block.1.layer.0.SelfAttention.
v.weight', 'encoder.block.1.layer.0.layer_norm.weight', 'encoder.block.1.layer.1.DenseReluDense.wi.weight', 'encoder.block.1.laye
r.1.DenseReluDense.wo.weight', 'encoder.block.1.layer.1.layer_norm.weight', 'encoder.block.10.layer.0.SelfAttention.k.weight', 'en
coder.block.10.layer.0.SelfAttention.o.weight', 'encoder.block.10.layer.0.SelfAttention.q.weight', 'encoder.block.10.layer.0.SelfA
ttention.v.weight', 'encoder.block.10.layer.0.layer_norm.weight', 'encoder.block.10.layer.1.DenseReluDense.wi.weight', 'encoder.bl
ock.10.layer.1.DenseReluDense.wo.weight', 'encoder.block.10.layer.1.layer_norm.weight', 'encoder.block.11.layer.0.SelfAttention.k.
weight', 'encoder.block.11.layer.0.SelfAttention.o.weight', 'encoder.block.11.layer.0.SelfAttention.q.weight', 'encoder.block.11.l
ayer.0.SelfAttention.v.weight', 'encoder.block.11.layer.0.layer_norm.weight', 'encoder.block.11.layer.1.DenseReluDense.wi.weight',
'encoder.block.11.layer.1.DenseReluDense.wo.weight', 'encoder.block.11.layer.1.layer_norm.weight', 'encoder.block.2.layer.0.SelfAt
tention.k.weight', 'encoder.block.2.layer.0.SelfAttention.o.weight', 'encoder.block.2.layer.0.SelfAttention.q.weight', 'encoder.bl
ock.2.layer.0.SelfAttention.v.weight', 'encoder.block.2.layer.0.layer_norm.weight', 'encoder.block.2.layer.1.DenseReluDense.wi.wei
ght', 'encoder.block.2.layer.1.DenseReluDense.wo.weight', 'encoder.block.2.layer.1.layer_norm.weight', 'encoder.block.3.layer.0.Se
lfAttention.k.weight', 'encoder.block.3.layer.0.SelfAttention.o.weight', 'encoder.block.3.layer.0.SelfAttention.q.weight', 'encode
r.block.3.layer.0.SelfAttention.v.weight', 'encoder.block.3.layer.0.layer_norm.weight', 'encoder.block.3.layer.1.DenseReluDense.w
i.weight', 'encoder.block.3.layer.1.DenseReluDense.wo.weight', 'encoder.block.3.layer.1.layer_norm.weight', 'encoder.block.4.laye
r.0.SelfAttention.k.weight', 'encoder.block.4.layer.0.SelfAttention.o.weight', 'encoder.block.4.layer.0.SelfAttention.q.weight',
'encoder.block.4.layer.0.SelfAttention.v.weight', 'encoder.block.4.layer.0.layer_norm.weight', 'encoder.block.4.layer.1.DenseReluD
ense.wi.weight', 'encoder.block.4.layer.1.DenseReluDense.wo.weight', 'encoder.block.4.layer.1.layer_norm.weight', 'encoder.block.
5.layer.0.SelfAttention.k.weight', 'encoder.block.5.layer.0.SelfAttention.o.weight', 'encoder.block.5.layer.0.SelfAttention.q.weig
ht', 'encoder.block.5.layer.0.SelfAttention.v.weight', 'encoder.block.5.layer.0.layer_norm.weight', 'encoder.block.5.layer.1.Dense
ReluDense.wi.weight', 'encoder.block.5.layer.1.DenseReluDense.wo.weight', 'encoder.block.5.layer.1.layer_norm.weight', 'encoder.bl
ock.6.layer.0.SelfAttention.k.weight', 'encoder.block.6.layer.0.SelfAttention.o.weight', 'encoder.block.6.layer.0.SelfAttention.q.
weight', 'encoder.block.6.layer.0.SelfAttention.v.weight', 'encoder.block.6.layer.0.layer_norm.weight', 'encoder.block.6.layer.1.D
enseReluDense.wi.weight', 'encoder.block.6.layer.1.DenseReluDense.wo.weight', 'encoder.block.6.layer.1.layer_norm.weight', 'encode
r.block.7.layer.0.SelfAttention.k.weight', 'encoder.block.7.layer.0.SelfAttention.o.weight', 'encoder.block.7.layer.0.SelfAttentio
n.q.weight', 'encoder.block.7.layer.0.SelfAttention.v.weight', 'encoder.block.7.layer.0.layer_norm.weight', 'encoder.block.7.laye
r.1.DenseReluDense.wi.weight', 'encoder.block.7.layer.1.DenseReluDense.wo.weight', 'encoder.block.7.layer.1.layer_norm.weight', 'e
ncoder.block.8.layer.0.SelfAttention.k.weight', 'encoder.block.8.layer.0.SelfAttention.o.weight', 'encoder.block.8.layer.0.SelfAtt
ention.q.weight', 'encoder.block.8.layer.0.SelfAttention.v.weight', 'encoder.block.8.layer.0.layer_norm.weight', 'encoder.block.8.
layer.1.DenseReluDense.wi.weight', 'encoder.block.8.layer.1.DenseReluDense.wo.weight', 'encoder.block.8.layer.1.layer_norm.weigh
t', 'encoder.block.9.layer.0.SelfAttention.k.weight', 'encoder.block.9.layer.0.SelfAttention.o.weight', 'encoder.block.9.layer.0.S
elfAttention.q.weight', 'encoder.block.9.layer.0.SelfAttention.v.weight', 'encoder.block.9.layer.0.layer_norm.weight', 'encoder.bl
ock.9.layer.1.DenseReluDense.wi.weight', 'encoder.block.9.layer.1.DenseReluDense.wo.weight', 'encoder.block.9.layer.1.layer_norm.w
eight', 'encoder.embed_tokens.weight', 'encoder.final_layer_norm.weight', 'shared.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [4]:
```python
import torch
from transformers import AutoTokenizer, AutoModel

# Load once globally (so it doesn't reload every call)
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
MODEL_NAME = "google/canine-s"  # or "google/canine-c" (larger, slower)

# Load model + tokenizer only once
tokenizer_canine = AutoTokenizer.from_pretrained(MODEL_NAME)
model_canine = AutoModel.from_pretrained(MODEL_NAME).to(DEVICE)
model_canine.eval()

@torch.no_grad()
def get_canine_embedding(url, max_length=50, pooling="mean"):
    """
    Returns a fixed-length CANINE embedding for a single URL string.

    Args:
        url (str): URL text to encode.
        max_length (int): Maximum length of character sequence (truncate/pad to this).
        pooling (str): 'mean' or 'max' for pooling token embeddings into one vector.

    Returns:
        np.ndarray: 1D embedding vector (shape [hidden_dim], e.g. [768]).
    """
    import numpy as np

    # Tokenize
    encoded = tokenizer_canine(
        url,
        truncation=True,
        padding="max_length",
        max_length=max_length,
        return_tensors="pt"
    ).to(DEVICE)

    # Forward pass
    outputs = model_canine(**encoded)
    hidden = outputs.last_hidden_state  # [1, seq_len, hidden_dim]
    mask = encoded["attention_mask"].unsqueeze(-1).type_as(hidden)  # [1, seq_len, 1]
```

```
    # Pooling
    if pooling == "mean":
        summed = (hidden * mask).sum(dim=1)
        denom = mask.sum(dim=1).clamp(min=1e-9)
        pooled = summed / denom  # [1, hidden_dim]
    elif pooling == "max":
        hidden = hidden.masked_fill(mask == 0, -1e9)
        pooled = hidden.max(dim=1).values  # [1, hidden_dim]
    else:
        raise ValueError("Pooling must be 'mean' or 'max'")

    return pooled.squeeze(0).cpu().numpy()  # (hidden_dim,)
```

In [36]:

```
# ============================================================
# Character Encoding Setup
# ============================================================

# Allowed printable ASCII chars
ascii_chars = [chr(i) for i in range(32, 127)]

# Special control tokens
special_tokens = [
    '<PAD>', '<UNK>',
]

# Build vocab and mapping
vocab = special_tokens + ascii_chars
char2idx = {ch: i for i, ch in enumerate(vocab)}



def encode(text, max_len=100):
    indices = torch.full((max_len,), char2idx["<PAD>"], dtype=torch.long)
    text = text.lower()[:max_len]
    for i, c in enumerate(text):
        indices[i] = char2idx.get(c, char2idx["<UNK>"])

    return indices
```

```
encoded_data = {}
frac = 1
gen = all_dataset()

x=0
for name, splits in  gen:
    encoded_data[name] = {}
    print(f"\n🔡 Encoding structured URLs for {name}...")

    for split_name, df in zip(['train', 'valid', 'test'], splits):
        df = df.sample(frac=frac, random_state=42)
        df["encode"] = df["url"].progress_apply(encode)
        encoded_data[name][split_name] = df
        print(f"  ✅ {split_name}: Encoded {len(df)} URLs")

    x+=1
    if x > 2:
        next(gen)

print("\n✅ All datasets encoded with proper start/end markers and padding!")
```

```
🔡 Encoding structured URLs for Dataset 1 (Malicious URLs)...
100%|██████████| 417732/417732 [01:17<00:00, 5364.14it/s]
  ✅ train: Encoded 417732 URLs
100%|██████████| 52217/52217 [00:09<00:00, 5434.72it/s]
  ✅ valid: Encoded 52217 URLs
100%|██████████| 52217/52217 [00:09<00:00, 5474.58it/s]
  ✅ test: Encoded 52217 URLs

🔡 Encoding structured URLs for Dataset 2 (ndarvind/phiusiil-phishing)...
100%|██████████| 188296/188296 [00:23<00:00, 8088.25it/s]
  ✅ train: Encoded 188296 URLs
100%|██████████| 23537/23537 [00:02<00:00, 7928.86it/s]
  ✅ valid: Encoded 23537 URLs
100%|██████████| 23537/23537 [00:02<00:00, 8152.26it/s]
```

```
✅ test: Encoded 23537 URLs

🔠 Encoding structured URLs for Dataset 3 (kmack/Phishing_urls)...
100%|██████████| 528097/528097 [01:24<00:00, 6280.96it/s]
✅ train: Encoded 528097 URLs
100%|██████████| 66012/66012 [00:10<00:00, 6308.68it/s]
✅ valid: Encoded 66012 URLs
100%|██████████| 66013/66013 [00:10<00:00, 6489.65it/s]
✅ test: Encoded 66013 URLs

✅ All datasets encoded with proper start/end markers and padding!
```

In [37]:

```python
from torch.utils.data import TensorDataset, DataLoader
# ============================================================
# Convert to TensorDataset and DataLoader
# ============================================================

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_size = 32

dataloader_dict = {}

def make_tensor_dataset(df):
    url_tensor = torch.stack(list(df["encode"]))
    labels_tensor = torch.tensor(df["label"].values, dtype=torch.long)
    return TensorDataset(url_tensor, labels_tensor)

for name, splits in encoded_data.items():
    dataloader_dict[name] = {}
    print(f"\n📦 Creating DataLoaders for {name}...")

    train_set = make_tensor_dataset(splits["train"])
    val_set = make_tensor_dataset(splits["valid"])
    test_set = make_tensor_dataset(splits["test"])

    dataloader_dict[name]["train_loader"] = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=
    dataloader_dict[name]["val_loader"] = DataLoader(val_set, batch_size=batch_size, shuffle=False, num_workers=4, pin_memory=Tru
    dataloader_dict[name]["test_loader"] = DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=4, pin_memory=T
```

```
    print(f"✅ DataLoaders ready for {name} (Train/Val/Test)")

print("\n🚀 All DataLoaders are ready in `dataloader_dict`!")

# Example Access:
# dataloader_dict["dataset1"]["train_loader"]
```

```
📦 Creating DataLoaders for Dataset 1 (Malicious URLs)...
✅ DataLoaders ready for Dataset 1 (Malicious URLs) (Train/Val/Test)

📦 Creating DataLoaders for Dataset 2 (ndarvind/phiusiil-phishing)...
✅ DataLoaders ready for Dataset 2 (ndarvind/phiusiil-phishing) (Train/Val/Test)

📦 Creating DataLoaders for Dataset 3 (kmack/Phishing_urls)...
✅ DataLoaders ready for Dataset 3 (kmack/Phishing_urls) (Train/Val/Test)

🚀 All DataLoaders are ready in `dataloader_dict`!
```

In [38]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# ================================================
#  🔹 TinyByT5 Encoder (Reduced depth for short sequences)
# ================================================
class TinyByT5Encoder(nn.Module):
    def __init__(self,
                 vocab_size=256,
                 d_model=128,
                 num_layers=2,
                 num_heads=2,
                 dim_ff=256,
                 max_len=100,
                 n_out=128,
                 dropout=0.1):
        super().__init__()

        #  🔹 Byte embedding layer (0–255)
        self.embedding = nn.Embedding(vocab_size, d_model)

        #  🔹 Positional embeddings
```

```
        self.pos_embedding = nn.Embedding(max_len, d_model)

        #  🔹 Transformer encoder layers
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model,
            nhead=num_heads,
            dim_feedforward=dim_ff,
            dropout=dropout,
            activation="gelu",
            batch_first=True,
            norm_first=True,
        )
        self.encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)

        #  🔹 Final normalization
        self.final_norm = nn.LayerNorm(d_model)
        self.projection = nn.Linear(in_features=d_model, out_features=n_out)

    def forward(self, x):
        """
        x: (batch, seq_len) — byte indices [0–255]
        """
        #batch_size, seq_len = x.size()
        #device = x.device

        #positions = torch.arange(seq_len, device=device).unsqueeze(0).expand(batch_size, -1)
        #x = self.embedding(x) + self.pos_embedding(positions)

        #x = self.encoder(x)
        x = self.embedding(x)
        x = self.projection(self.final_norm(x))

        return x  # (B, L, d_model)
```

In [39]:
```
class SEBlock(nn.Module):
    def __init__(self, channels, reduction=8):
        super(SEBlock, self).__init__()
        self.fc1 = nn.Linear(channels, channels // reduction)
        self.fc2 = nn.Linear(channels // reduction, channels)
        self.sigmoid = nn.Sigmoid()
```

```
    def forward(self, x):
        # x: (B, C, L)
        w = x.mean(dim=2)                       # Global Average Pooling -> (B, C)
        w = F.relu(self.fc1(w))
        w = self.sigmoid(self.fc2(w))
        w = w.unsqueeze(2)                      # (B, C, 1)
        return x * w                            # scale features
```

In [40]:
```
#  🔹 Residual Depthwise-Separable Multi-Kernel Block
class ResidualConvBlockDW(nn.Module):
    def __init__(self, in_ch, out_ch, kernel_sizes=[3, 5, 7], reduction=16):
        super().__init__()
        self.branches = nn.ModuleList()

        mid_ch = max(in_ch // 2, 8)  # reduce dimension before heavy convs

        for k in kernel_sizes:
            branch = nn.Sequential(
                # (B) Reduce channels first
                nn.Conv1d(in_ch, mid_ch, kernel_size=1, bias=False),
                #nn.BatchNorm1d(mid_ch),
                nn.ReLU(inplace=True),

                # (A) Depthwise conv
                nn.Conv1d(mid_ch, mid_ch, kernel_size=k, padding=k // 2, groups=mid_ch, bias=False),
                #nn.BatchNorm1d(mid_ch),
                nn.ReLU(inplace=True),

                # Pointwise to expand to out_ch
                nn.Conv1d(mid_ch, out_ch, kernel_size=1, bias=False),
                #nn.BatchNorm1d(out_ch),
                nn.ReLU(inplace=True)
            )
            self.branches.append(branch)

        # Combine all kernel branches
        self.merge_conv = nn.Conv1d(out_ch * len(kernel_sizes), out_ch, kernel_size=1, bias=False)
        #self.merge_bn = nn.BatchNorm1d(out_ch)
```

```
        self.se = SEBlock(out_ch, reduction)
        self.shortcut = nn.Conv1d(in_ch, out_ch, kernel_size=1) if in_ch != out_ch else nn.Identity()

    def forward(self, x):
        # Parallel multi-kernel branches
        out = [branch(x) for branch in self.branches]
        out = torch.cat(out, dim=1)

        out = self.merge_conv(out)
        #out = self.merge_bn(out)
        out = self.se(out)
        out += self.shortcut(x)
        return F.relu(out)
```

In [41]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class URLBinaryCNN(nn.Module):
    def __init__(self, vocab_size, embed_dim=128, maxlen=100):
        super(URLBinaryCNN, self).__init__()
        self.maxlen = maxlen

        self.transformer = TinyByT5Encoder(
            vocab_size=vocab_size,
            max_len=maxlen,
            d_model=512,
            n_out=embed_dim
        )
        #  🔹 1st Conv block
        self.conv_layer1 = ResidualConvBlockDW(embed_dim, 128)
        self.conv_layer2 = ResidualConvBlockDW(64, 32)


         #  🔹 2st Conv block
        self.conv2_3x3 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.convreduce2_1x1 = nn.Conv1d(in_channels=128, out_channels=1, kernel_size=3, padding=1)
```

```
        self.conv2_5x5 = nn.Conv1d(in_channels=1, out_channels=64, kernel_size=5, padding=2)
        self.conv2_1x1 = nn.Conv1d(in_channels=64*2, out_channels=64, kernel_size=3, padding=1)
        self.layer_norm2 = nn.LayerNorm(normalized_shape=[64, maxlen])


         #  🔹 3st Conv block
        self.conv3_3x3 = nn.Conv1d(in_channels=64, out_channels=32, kernel_size=3, padding=1)
        self.convreduce3_1x1 = nn.Conv1d(in_channels=64, out_channels=1, kernel_size=1, padding=0)
        self.conv3_5x5 = nn.Conv1d(in_channels=64, out_channels=32, kernel_size=5, padding=2)
        self.conv3_7 = nn.Conv1d(in_channels=64, out_channels=32, kernel_size=7, padding=3)
        self.conv3_1x1 = nn.Conv1d(in_channels=32*3, out_channels=32, kernel_size=1, padding=0)
        self.layer_norm3 = nn.LayerNorm(normalized_shape=[32, maxlen])

        self.lstm = nn.LSTM(input_size=128, hidden_size=128, num_layers=1,
                            batch_first=True)
        self.bilstm = nn.LSTM(input_size=128, hidden_size=128, num_layers=1,
                            batch_first=True, bidirectional=True)
        #self.aap = nn.AdaptiveAvgPool1d(1)

        #  🔹 Fully connected layers
        self.layer_norm = nn.LayerNorm(128 * 2)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128*2, 128)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self.fc_layers = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.5),



            nn.Linear(64, 1)
        )
    def forward(self, x):
        # (batch, seq_len)
        x = self.transformer(x)           # (B, L, E)

        x = x.permute(0, 2, 1)            # (B, E, L)
```

```
        #  🔹 Block 1
        #x3 = F.relu(self.conv1_3x3(x))

        #xr = F.relu(self.convreduce1_1x1(x))
        #x5 = F.relu(self.conv1_5x5(x))
        #x7 = F.relu(self.conv1_7(x))

        #x = torch.cat([x3, x5, x7], dim=1)
        #print(x.shape)
        #x = F.relu(self.conv1_1x1(x))

        #x = self.layer_norm1(x)


        '''
        #  🔹 Block 2
        x3 = F.relu(self.conv2_3x3(x))
        xr = F.relu(self.convreduce2_1x1(x))
        x5 = F.relu(self.conv2_5x5(xr))
        x = torch.cat([x3, x5], dim=1)
        x = F.relu(self.conv2_1x1(x))
        x = self.layer_norm2(x)
        '''
        #  🔹 Block 3
        #x = F.relu(self.conv3_3x3(x))
        #xr = F.relu(self.convreduce3_1x1(x))
        #x5 = F.relu(self.conv3_5x5(x))
        #x7 = F.relu(self.conv3_7(x))
        #x = torch.cat([x3, x5, x7], dim=1)
        #x = F.relu(self.conv3_1x1(x))
        #x = self.layer_norm3(x)
        x = self.conv_layer1(x)
        #x = self.conv_layer2(x)
        x = x.permute(0, 2, 1)

        x, _ = self.bilstm(x)
        x = self.layer_norm(x)
        x = x[:, -1, :]
        #  🔹 Global Average Pooling + FC
```

```
        #x = self.aap(x)                    # (batch, channels, 1)
        x = self.flatten(x)                 # (batch, channels)
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc_layers(x)

        return torch.sigmoid(x)

    def extract_features(self, x):
        """Return deep features before final FC layers."""
        x = self.transformer(x)
        x = x.permute(0, 2, 1)
        x = self.conv_layer1(x)
        x = x.permute(0, 2, 1)
        x, _ = self.bilstm(x)
        x = x[:, -1, :]          # shape: [batch, 32]
        x = self.flatten(x)
        x = self.dropout(self.relu(self.fc1(x)))
        return x
```

In [42]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import sys

class Train:
    def __init__(self,
                 model,
                 criterion,
                 transformer_optimizer=None,
                 main_optimizer = None,
                 scheduler_t=None,
                 scheduler_c=None,
                 train_loader=None,
                 val_loader=None,
                 device=torch.device("cuda" if torch.cuda.is_available() else "cpu")):

        """
        optimizer_groups: dict with keys like {"transformer": optimizer1, "cnn": optimizer2}
        schedulers: dict with keys matching optimizer_groups (optional)
        """
```

```
        self.model = model
        self.criterion = criterion

        self.transformer_params = []
        self.cnn_params = []
        for name, param in model.named_parameters():
            if name.startswith("transformer."):
                self.transformer_params.append(param)
            else:
                self.cnn_params.append(param)



        self.transformer_optimizer = optim.Adam(self.transformer_params, lr=1e-4) if transformer_optimizer is None  else transfor
        self.main_optimizer = optim.Adam(self.cnn_params, lr=1e-3) if main_optimizer is None else main_optimizer
        self.scheduler_t = optim.lr_scheduler.ReduceLROnPlateau(self.transformer_optimizer, mode='min', factor=0.5, patience=2) i
        self.scheduler_c = optim.lr_scheduler.ReduceLROnPlateau(self.main_optimizer, mode='min', factor=0.5, patience=2) if sched
        self.device = device
        self.train_loader = train_loader
        self.val_loader = val_loader
        self.train_losses, self.val_losses = [], []
        self.train_accs, self.val_accs = [], []


    def freeze_module(self, module):
        for param in module.parameters():
            param.requires_grad = False

    def unfreeze_module(self, module):
        for param in module.parameters():
            param.requires_grad = True


    def train(self, epochs_list=[3,3,4], early_stopping=True, frac=1.0, val_frac=1.0, alt_cycle = 2,  log=0):
        for phase, epochs in enumerate(epochs_list):
            for epoch in range(epochs):
                phase=3
                if phase == 0:
                    # 🧠 Train Transformer — freeze CNN/LSTM layers
                    self.unfreeze_module(self.model.transformer)
                    for name, module in self.model.named_children():
```

```
                    if name != "transformer":
                        self.freeze_module(module)
                    else:
                        self.unfreeze_module(module)
                active_optims = [self.transformer_optimizer]
                active_scheds = [self.scheduler_t]
                phase_name = "Transformer"
            elif phase == 1:
                # 🎯 Train CNN/LSTM/FC — freeze Transformer
                self.freeze_module(self.model.transformer)
                for name, module in self.model.named_children():
                    if name == "transformer":
                        self.freeze_module(module)
                    else:
                        self.unfreeze_module(module)
                active_optims = [self.main_optimizer]
                active_scheds = [self.scheduler_c]
                phase_name = "CNN"
            else:
                self.freeze_module(self.model.transformer)
                for name, module in self.model.named_children():
                    self.unfreeze_module(module)
                active_optims = [self.transformer_optimizer, self.main_optimizer]
                active_scheds = [self.scheduler_t, self.scheduler_c]
                phase_name = "CNN + transformer"



            self.model.train()
            train_loss, correct_train, total_train = 0, 0, 0
            max_batches = int(len(self.train_loader) * frac)

            for batch_idx, (batch_x, batch_y) in enumerate(self.train_loader):
                if batch_idx >= max_batches:
                    break

                batch_x, batch_y = batch_x.to(self.device, non_blocking=True), batch_y.to(self.device, non_blocking=True).flo

                for opt in active_optims:
                    opt.zero_grad()
```

```
                    outputs = self.model(batch_x)
                    loss = self.criterion(outputs, batch_y)
                    loss.backward()

                    for opt in active_optims:
                        opt.step()




                    # === Metrics ===
                    batch_loss = loss.item()
                    preds = (outputs >= 0.5).float()
                    batch_acc = (preds == batch_y).float().mean().item()

                    if log >= 1 and (batch_idx + 1) % (20/log) == 0:
                        print(f"\rEpoch {epoch+1}/{epochs}: Training {phase_name} | "
                            f"Batch {batch_idx+1}/{max_batches} | "
                            f"Loss: {batch_loss:.4f}, Acc: {batch_acc:.4f}", end='')


                if log >= 2:
                    max_batches = max(int(len(self.train_loader) * frac), 1)
                    print(f'\r total training batch size {max_batches}'.ljust(100), end='')
                    with torch.no_grad():
                        for batch_idx, (batch_x, batch_y) in enumerate(self.train_loader):
                            if batch_idx >= max_batches:
                                break

                            batch_x, batch_y = batch_x.to(self.device, non_blocking=True), batch_y.to(self.device, non_blocking=T

                            outputs = self.model(batch_x)
                            loss = self.criterion(outputs, batch_y)

                            batch_loss = loss.item()
                            preds = (outputs >= 0.5).float()
                            batch_acc = (preds == batch_y).float().mean().item()

                            train_loss += batch_loss * batch_x.size(0)
                            correct_train += (preds == batch_y).sum().item()
                            total_train += batch_x.size(0)
```

```
                    avg_train_loss = train_loss / total_train
                    train_acc = correct_train / total_train
                    self.train_losses.append(avg_train_loss)
                    self.train_accs.append(train_acc)


                # === Validation ===
                if self.val_loader is not None:
                    avg_val_loss, val_acc = self.evaluate(val_frac)
                    self.val_losses.append(avg_val_loss)
                    self.val_accs.append(val_acc)

                    print(f"\rEpoch {epoch+1}/{epochs} Training {phase_name}| "
                        f"Train Loss: {avg_train_loss:.4f}, Train Acc: {train_acc:.4f} | "
                        f"Val Loss: {avg_val_loss:.4f}, Val Acc: {val_acc:.4f}")
                else:
                    print(f"\rEpoch {epoch+1}/{epochs} Training {phase_name}| "
                        f"Train Loss: {avg_train_loss:.4f}, Train Acc: {train_acc:.4f}")


            for sched in active_scheds:
                sched.step(avg_val_loss)



def evaluate(self, frac=1.0):
    self.model.eval()
    val_loss, correct_val, total_val = 0, 0, 0
    max_batches = max(int(len(self.val_loader) * frac), 1)
    print(f'\r total validation batch size {max_batches}'.ljust(100), end='')
    with torch.no_grad():
        for batch_idx, (batch_x, batch_y) in enumerate(self.val_loader):
            if batch_idx >= max_batches:
                break
            batch_x, batch_y = batch_x.to(self.device, non_blocking=True), batch_y.to(self.device, non_blocking=True).float()
            outputs = self.model(batch_x)
            loss = self.criterion(outputs, batch_y)
            avg_batch_loss = loss.item()
            val_loss += avg_batch_loss * batch_x.size(0)
            preds = (outputs >= 0.5).float()
```

```
                correct_val += (preds == batch_y).sum().item()
                total_val += batch_x.size(0)

        avg_val_loss = val_loss / total_val
        val_acc = correct_val / total_val
        return avg_val_loss, val_acc
```

In [43]:

```
import sys
import torch
import torch.nn as nn
import torch.nn.functional as F
from tqdm import tqdm
from lion_pytorch import Lion
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ============================================================
#  🦴 Training Config
# ============================================================
num_epochs = [1,1,1]
lr = 0.01

# Store all dataset metrics
all_results = {}

# ============================================================
# 🔁 Training Loop for Each Dataset
# ============================================================
nn_model = {}
for dataset_name, loaders in dataloader_dict.items():
    print("\n" + "="*70)
    print(f"🚀 Training model on {dataset_name.upper()} dataset")
    print("="*70)
    for frac in  [1.0]:
        print(f"🧩 Using {frac*100:.0f}% of training data".center(50, '_'))
        train_loader = loaders["train_loader"]
        val_loader = loaders["val_loader"]

        # Initialize model, loss, optimizer
        nn_model[dataset_name] = URLBinaryCNN(vocab_size=len(vocab)).to(device)
```

```
        criterion = nn.BCELoss()
        optimizer = torch.optim.Adam(nn_model[dataset_name].parameters(), lr=lr, weight_decay=lr/10)
        trainer = Train(nn_model[dataset_name], criterion, train_loader=train_loader, val_loader=val_loader)
        # Lists to track performance
        trainer.train(num_epochs,lr,frac=frac,val_frac=frac, log=2)



    # Store all results for this dataset
    all_results[dataset_name] = {
        "train_losses": trainer.train_losses,
        "val_losses": trainer.val_losses,
        "train_accs": trainer.train_accs,
        "val_accs": trainer.val_accs,
        "final_val_acc": trainer.val_accs[-1],
        "final_val_loss": trainer.val_losses[-1]
    }

print("\n✅ All datasets trained successfully!")

# ============================================================
# 📊 Summary of All Results
# ============================================================
print("\n" + "="*70)
print("📈 Final Validation Accuracy Summary")
print("="*70)
for name, res in all_results.items():
    print(f"{name:<20} | Val Acc: {res['final_val_acc']:.4f} | Val Loss: {res['final_val_loss']:.4f}")
```

```
======================================================================
🚀 Training model on DATASET 1 (MALICIOUS URLS) dataset
======================================================================
__________🧩 Using 100% of training data___________
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\torch\nn\modules\transformer.py:392: UserWarning: enable_
nested_tensor is True, but self.use_nested_tensor is False because encoder_layer.norm_first was True
  warnings.warn(
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\torch\utils\data\dataloader.py:666: UserWarning: 'pin_mem
ory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
  warnings.warn(warn_msg)
```

```
Epoch 1/1 Training CNN + transformer| Train Loss: 0.0912, Train Acc: 0.9731 | Val Loss: 0.0886, Val Acc: 0.9727
Epoch 1/1 Training CNN + transformer| Train Loss: 0.0705, Train Acc: 0.9776 | Val Loss: 0.0700, Val Acc: 0.9768
Epoch 1/1 Training CNN + transformer| Train Loss: 0.0623, Train Acc: 0.9810 | Val Loss: 0.0614, Val Acc: 0.9805

======================================================================
🚀 Training model on DATASET 2 (NDARVIND/PHIUSIIL-PHISHING) dataset
======================================================================
__________🧩 Using 100% of training data__________
Epoch 1/1 Training CNN + transformer| Train Loss: 0.0161, Train Acc: 0.9976 | Val Loss: 0.0157, Val Acc: 0.9974
Epoch 1/1 Training CNN + transformer| Train Loss: 0.0149, Train Acc: 0.9979 | Val Loss: 0.0139, Val Acc: 0.9977
Epoch 1/1 Training CNN + transformer| Train Loss: 0.0147, Train Acc: 0.9980 | Val Loss: 0.0157, Val Acc: 0.9977

======================================================================
🚀 Training model on DATASET 3 (KMACK/PHISHING_URLS) dataset
======================================================================
__________🧩 Using 100% of training data__________
Epoch 1/1 Training CNN + transformer| Train Loss: 0.2989, Train Acc: 0.8804 | Val Loss: 0.2861, Val Acc: 0.8802
Epoch 1/1 Training CNN + transformer| Train Loss: 0.2346, Train Acc: 0.9077 | Val Loss: 0.2349, Val Acc: 0.9053
Epoch 1/1 Training CNN + transformer| Train Loss: 0.2327, Train Acc: 0.9086 | Val Loss: 0.2300, Val Acc: 0.9065

✅ All datasets trained successfully!

======================================================================
📈 Final Validation Accuracy Summary
======================================================================
Dataset 1 (Malicious URLs) | Val Acc: 0.9805 | Val Loss: 0.0614
Dataset 2 (ndarvind/phiusiil-phishing) | Val Acc: 0.9977 | Val Loss: 0.0157
Dataset 3 (kmack/Phishing_urls) | Val Acc: 0.9065 | Val Loss: 0.2300
```

In [44]:
```
torch.save(model.state_dict(), "url_cnn_lstm.pth")
```

In [45]:
```
#xgbost model + cnn based dl model
def extract_features(self, x):
        """Return deep features before final FC layers."""
        x = self.transformer(x)
        x = x.permute(0, 2, 1)
        x3 = F.relu(self.conv1_3x3(x))
        x5 = F.relu(self.conv1_5x5(x))
        x7 = F.relu(self.conv1_7(x))
        x = torch.cat([x3, x5, x7], dim=1)
```

```
        x = F.relu(self.conv1_1x1(x))
        x = x.permute(0, 2, 1)
        x, _ = self.lstm(x)
        x = x[:, -1, :]          # shape: [batch, 32]
        return x
```

In [46]:

```
import numpy as np
from tqdm import tqdm
import torch

model.eval()
features, labels = [], []

#  🔹 Extract CNN features for training set
with torch.no_grad():
    for x_batch, y_batch in tqdm(train_loader, desc="🔍 Extracting Train Features", unit="batch"):
        x_batch = x_batch.to(device, non_blocking=True)
        feats = nn_model[dataset_name].extract_features(x_batch)
        features.append(feats.cpu().numpy())
        labels.append(y_batch.cpu().numpy())

X_train = np.concatenate(features, axis=0)
y_train = np.concatenate(labels, axis=0)

# Free memory before val extraction
del features, labels, x_batch, y_batch, feats
torch.cuda.empty_cache()

#  🔹 Extract CNN features for validation set
features, labels = [], []
with torch.no_grad():
    for x_batch, y_batch in tqdm(val_loader, desc="🔍 Extracting Val Features", unit="batch"):
        x_batch = x_batch.to(device, non_blocking=True)
        feats = nn_model[dataset_name].extract_features(x_batch)
        features.append(feats.cpu().numpy())
        labels.append(y_batch.cpu().numpy())

X_val = np.concatenate(features, axis=0)
y_val = np.concatenate(labels, axis=0)
```

```
🔍 Extracting Train Features: 100%|██████████| 16504/16504 [04:56<00:00, 55.76batch/s]
🔍 Extracting Val Features: 100%|██████████| 2063/2063 [00:39<00:00, 52.13batch/s]
```

In [47]:
```python
import xgboost as xgb

xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    learning_rate=0.01,
    eval_metric="logloss",
    max_depth = 10,
    random_state=42
)

xgb_model.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    verbose=True
)
```

```
[0]     validation_0-logloss:0.68590
[1]     validation_0-logloss:0.67879
[2]     validation_0-logloss:0.67182
[3]     validation_0-logloss:0.66499
[4]     validation_0-logloss:0.65829
[5]     validation_0-logloss:0.65172
[6]     validation_0-logloss:0.64528
[7]     validation_0-logloss:0.63895
[8]     validation_0-logloss:0.63275
[9]     validation_0-logloss:0.62666
[10]    validation_0-logloss:0.62068
[11]    validation_0-logloss:0.61481
[12]    validation_0-logloss:0.60905
[13]    validation_0-logloss:0.60339
[14]    validation_0-logloss:0.59784
[15]    validation_0-logloss:0.59238
[16]    validation_0-logloss:0.58701
[17]    validation_0-logloss:0.58174
[18]    validation_0-logloss:0.57656
[19]    validation_0-logloss:0.57148
[20]    validation_0-logloss:0.56648
[21]    validation_0-logloss:0.56156
[22]    validation_0-logloss:0.55673
[23]    validation_0-logloss:0.55198
[24]    validation_0-logloss:0.54731
[25]    validation_0-logloss:0.54272
[26]    validation_0-logloss:0.53821
[27]    validation_0-logloss:0.53377
[28]    validation_0-logloss:0.52941
[29]    validation_0-logloss:0.52512
[30]    validation_0-logloss:0.52090
[31]    validation_0-logloss:0.51674
[32]    validation_0-logloss:0.51265
[33]    validation_0-logloss:0.50863
[34]    validation_0-logloss:0.50468
[35]    validation_0-logloss:0.50079
[36]    validation_0-logloss:0.49696
[37]    validation_0-logloss:0.49318
[38]    validation_0-logloss:0.48948
[39]    validation_0-logloss:0.48582
[40]    validation_0-logloss:0.48223
```

```
[41]    validation_0-logloss:0.47868
[42]    validation_0-logloss:0.47520
[43]    validation_0-logloss:0.47177
[44]    validation_0-logloss:0.46839
[45]    validation_0-logloss:0.46507
[46]    validation_0-logloss:0.46179
[47]    validation_0-logloss:0.45856
[48]    validation_0-logloss:0.45539
[49]    validation_0-logloss:0.45226
[50]    validation_0-logloss:0.44918
[51]    validation_0-logloss:0.44614
[52]    validation_0-logloss:0.44316
[53]    validation_0-logloss:0.44021
[54]    validation_0-logloss:0.43730
[55]    validation_0-logloss:0.43444
[56]    validation_0-logloss:0.43163
[57]    validation_0-logloss:0.42885
[58]    validation_0-logloss:0.42611
[59]    validation_0-logloss:0.42342
[60]    validation_0-logloss:0.42076
[61]    validation_0-logloss:0.41814
[62]    validation_0-logloss:0.41556
[63]    validation_0-logloss:0.41302
[64]    validation_0-logloss:0.41052
[65]    validation_0-logloss:0.40805
[66]    validation_0-logloss:0.40562
[67]    validation_0-logloss:0.40322
[68]    validation_0-logloss:0.40085
[69]    validation_0-logloss:0.39853
[70]    validation_0-logloss:0.39623
[71]    validation_0-logloss:0.39396
[72]    validation_0-logloss:0.39173
[73]    validation_0-logloss:0.38952
[74]    validation_0-logloss:0.38736
[75]    validation_0-logloss:0.38521
[76]    validation_0-logloss:0.38309
[77]    validation_0-logloss:0.38101
[78]    validation_0-logloss:0.37895
[79]    validation_0-logloss:0.37692
[80]    validation_0-logloss:0.37492
[81]    validation_0-logloss:0.37295
```

```
[82]    validation_0-logloss:0.37100
[83]    validation_0-logloss:0.36908
[84]    validation_0-logloss:0.36719
[85]    validation_0-logloss:0.36532
[86]    validation_0-logloss:0.36348
[87]    validation_0-logloss:0.36167
[88]    validation_0-logloss:0.35988
[89]    validation_0-logloss:0.35811
[90]    validation_0-logloss:0.35636
[91]    validation_0-logloss:0.35464
[92]    validation_0-logloss:0.35295
[93]    validation_0-logloss:0.35127
[94]    validation_0-logloss:0.34962
[95]    validation_0-logloss:0.34800
[96]    validation_0-logloss:0.34640
[97]    validation_0-logloss:0.34482
[98]    validation_0-logloss:0.34325
[99]    validation_0-logloss:0.34171
[100]   validation_0-logloss:0.34019
[101]   validation_0-logloss:0.33869
[102]   validation_0-logloss:0.33721
[103]   validation_0-logloss:0.33574
[104]   validation_0-logloss:0.33430
[105]   validation_0-logloss:0.33289
[106]   validation_0-logloss:0.33148
[107]   validation_0-logloss:0.33009
[108]   validation_0-logloss:0.32873
[109]   validation_0-logloss:0.32737
[110]   validation_0-logloss:0.32604
[111]   validation_0-logloss:0.32473
[112]   validation_0-logloss:0.32343
[113]   validation_0-logloss:0.32214
[114]   validation_0-logloss:0.32088
[115]   validation_0-logloss:0.31963
[116]   validation_0-logloss:0.31839
[117]   validation_0-logloss:0.31718
[118]   validation_0-logloss:0.31598
[119]   validation_0-logloss:0.31479
[120]   validation_0-logloss:0.31362
[121]   validation_0-logloss:0.31247
[122]   validation_0-logloss:0.31133
```

```
[123]   validation_0-logloss:0.31021
[124]   validation_0-logloss:0.30910
[125]   validation_0-logloss:0.30800
[126]   validation_0-logloss:0.30691
[127]   validation_0-logloss:0.30584
[128]   validation_0-logloss:0.30479
[129]   validation_0-logloss:0.30374
[130]   validation_0-logloss:0.30272
[131]   validation_0-logloss:0.30170
[132]   validation_0-logloss:0.30070
[133]   validation_0-logloss:0.29971
[134]   validation_0-logloss:0.29873
[135]   validation_0-logloss:0.29777
[136]   validation_0-logloss:0.29682
[137]   validation_0-logloss:0.29588
[138]   validation_0-logloss:0.29495
[139]   validation_0-logloss:0.29403
[140]   validation_0-logloss:0.29313
[141]   validation_0-logloss:0.29223
[142]   validation_0-logloss:0.29135
[143]   validation_0-logloss:0.29048
[144]   validation_0-logloss:0.28962
[145]   validation_0-logloss:0.28877
[146]   validation_0-logloss:0.28794
[147]   validation_0-logloss:0.28711
[148]   validation_0-logloss:0.28629
[149]   validation_0-logloss:0.28549
[150]   validation_0-logloss:0.28469
[151]   validation_0-logloss:0.28390
[152]   validation_0-logloss:0.28312
[153]   validation_0-logloss:0.28235
[154]   validation_0-logloss:0.28159
[155]   validation_0-logloss:0.28084
[156]   validation_0-logloss:0.28010
[157]   validation_0-logloss:0.27936
[158]   validation_0-logloss:0.27864
[159]   validation_0-logloss:0.27792
[160]   validation_0-logloss:0.27722
[161]   validation_0-logloss:0.27652
[162]   validation_0-logloss:0.27583
[163]   validation_0-logloss:0.27515
```

```
[164]   validation_0-logloss:0.27448
[165]   validation_0-logloss:0.27382
[166]   validation_0-logloss:0.27316
[167]   validation_0-logloss:0.27252
[168]   validation_0-logloss:0.27188
[169]   validation_0-logloss:0.27125
[170]   validation_0-logloss:0.27063
[171]   validation_0-logloss:0.27001
[172]   validation_0-logloss:0.26941
[173]   validation_0-logloss:0.26881
[174]   validation_0-logloss:0.26822
[175]   validation_0-logloss:0.26763
[176]   validation_0-logloss:0.26705
[177]   validation_0-logloss:0.26648
[178]   validation_0-logloss:0.26592
[179]   validation_0-logloss:0.26536
[180]   validation_0-logloss:0.26481
[181]   validation_0-logloss:0.26426
[182]   validation_0-logloss:0.26372
[183]   validation_0-logloss:0.26319
[184]   validation_0-logloss:0.26266
[185]   validation_0-logloss:0.26214
[186]   validation_0-logloss:0.26163
[187]   validation_0-logloss:0.26113
[188]   validation_0-logloss:0.26063
[189]   validation_0-logloss:0.26014
[190]   validation_0-logloss:0.25965
[191]   validation_0-logloss:0.25917
[192]   validation_0-logloss:0.25869
[193]   validation_0-logloss:0.25822
[194]   validation_0-logloss:0.25776
[195]   validation_0-logloss:0.25730
[196]   validation_0-logloss:0.25685
[197]   validation_0-logloss:0.25641
[198]   validation_0-logloss:0.25598
[199]   validation_0-logloss:0.25555
[200]   validation_0-logloss:0.25513
[201]   validation_0-logloss:0.25471
[202]   validation_0-logloss:0.25430
[203]   validation_0-logloss:0.25388
[204]   validation_0-logloss:0.25347
```

```
[205]   validation_0-logloss:0.25307
[206]   validation_0-logloss:0.25267
[207]   validation_0-logloss:0.25228
[208]   validation_0-logloss:0.25190
[209]   validation_0-logloss:0.25152
[210]   validation_0-logloss:0.25114
[211]   validation_0-logloss:0.25076
[212]   validation_0-logloss:0.25039
[213]   validation_0-logloss:0.25002
[214]   validation_0-logloss:0.24966
[215]   validation_0-logloss:0.24931
[216]   validation_0-logloss:0.24895
[217]   validation_0-logloss:0.24860
[218]   validation_0-logloss:0.24826
[219]   validation_0-logloss:0.24792
[220]   validation_0-logloss:0.24759
[221]   validation_0-logloss:0.24726
[222]   validation_0-logloss:0.24694
[223]   validation_0-logloss:0.24662
[224]   validation_0-logloss:0.24630
[225]   validation_0-logloss:0.24598
[226]   validation_0-logloss:0.24568
[227]   validation_0-logloss:0.24537
[228]   validation_0-logloss:0.24507
[229]   validation_0-logloss:0.24477
[230]   validation_0-logloss:0.24448
[231]   validation_0-logloss:0.24418
[232]   validation_0-logloss:0.24389
[233]   validation_0-logloss:0.24361
[234]   validation_0-logloss:0.24333
[235]   validation_0-logloss:0.24305
[236]   validation_0-logloss:0.24278
[237]   validation_0-logloss:0.24251
[238]   validation_0-logloss:0.24224
[239]   validation_0-logloss:0.24197
[240]   validation_0-logloss:0.24171
[241]   validation_0-logloss:0.24145
[242]   validation_0-logloss:0.24120
[243]   validation_0-logloss:0.24095
[244]   validation_0-logloss:0.24070
[245]   validation_0-logloss:0.24045
```

```
[246]   validation_0-logloss:0.24020
[247]   validation_0-logloss:0.23996
[248]   validation_0-logloss:0.23973
[249]   validation_0-logloss:0.23950
[250]   validation_0-logloss:0.23927
[251]   validation_0-logloss:0.23904
[252]   validation_0-logloss:0.23882
[253]   validation_0-logloss:0.23859
[254]   validation_0-logloss:0.23838
[255]   validation_0-logloss:0.23816
[256]   validation_0-logloss:0.23795
[257]   validation_0-logloss:0.23773
[258]   validation_0-logloss:0.23752
[259]   validation_0-logloss:0.23732
[260]   validation_0-logloss:0.23711
[261]   validation_0-logloss:0.23691
[262]   validation_0-logloss:0.23671
[263]   validation_0-logloss:0.23652
[264]   validation_0-logloss:0.23633
[265]   validation_0-logloss:0.23613
[266]   validation_0-logloss:0.23594
[267]   validation_0-logloss:0.23575
[268]   validation_0-logloss:0.23557
[269]   validation_0-logloss:0.23539
[270]   validation_0-logloss:0.23521
[271]   validation_0-logloss:0.23503
[272]   validation_0-logloss:0.23485
[273]   validation_0-logloss:0.23468
[274]   validation_0-logloss:0.23451
[275]   validation_0-logloss:0.23434
[276]   validation_0-logloss:0.23418
[277]   validation_0-logloss:0.23401
[278]   validation_0-logloss:0.23385
[279]   validation_0-logloss:0.23370
[280]   validation_0-logloss:0.23354
[281]   validation_0-logloss:0.23339
[282]   validation_0-logloss:0.23324
[283]   validation_0-logloss:0.23308
[284]   validation_0-logloss:0.23293
[285]   validation_0-logloss:0.23278
[286]   validation_0-logloss:0.23264
```

```
[287]   validation_0-logloss:0.23249
[288]   validation_0-logloss:0.23234
[289]   validation_0-logloss:0.23219
[290]   validation_0-logloss:0.23205
[291]   validation_0-logloss:0.23191
[292]   validation_0-logloss:0.23177
[293]   validation_0-logloss:0.23164
[294]   validation_0-logloss:0.23150
[295]   validation_0-logloss:0.23137
[296]   validation_0-logloss:0.23124
[297]   validation_0-logloss:0.23111
[298]   validation_0-logloss:0.23098
[299]   validation_0-logloss:0.23086
[300]   validation_0-logloss:0.23073
[301]   validation_0-logloss:0.23061
[302]   validation_0-logloss:0.23049
[303]   validation_0-logloss:0.23037
[304]   validation_0-logloss:0.23025
[305]   validation_0-logloss:0.23014
[306]   validation_0-logloss:0.23002
[307]   validation_0-logloss:0.22991
[308]   validation_0-logloss:0.22980
[309]   validation_0-logloss:0.22969
[310]   validation_0-logloss:0.22958
[311]   validation_0-logloss:0.22947
[312]   validation_0-logloss:0.22936
[313]   validation_0-logloss:0.22926
[314]   validation_0-logloss:0.22916
[315]   validation_0-logloss:0.22905
[316]   validation_0-logloss:0.22895
[317]   validation_0-logloss:0.22885
[318]   validation_0-logloss:0.22876
[319]   validation_0-logloss:0.22866
[320]   validation_0-logloss:0.22856
[321]   validation_0-logloss:0.22847
[322]   validation_0-logloss:0.22838
[323]   validation_0-logloss:0.22829
[324]   validation_0-logloss:0.22820
[325]   validation_0-logloss:0.22811
[326]   validation_0-logloss:0.22802
[327]   validation_0-logloss:0.22794
```

```
[328]   validation_0-logloss:0.22785
[329]   validation_0-logloss:0.22776
[330]   validation_0-logloss:0.22768
[331]   validation_0-logloss:0.22759
[332]   validation_0-logloss:0.22752
[333]   validation_0-logloss:0.22744
[334]   validation_0-logloss:0.22735
[335]   validation_0-logloss:0.22727
[336]   validation_0-logloss:0.22719
[337]   validation_0-logloss:0.22711
[338]   validation_0-logloss:0.22703
[339]   validation_0-logloss:0.22695
[340]   validation_0-logloss:0.22688
[341]   validation_0-logloss:0.22680
[342]   validation_0-logloss:0.22672
[343]   validation_0-logloss:0.22665
[344]   validation_0-logloss:0.22657
[345]   validation_0-logloss:0.22650
[346]   validation_0-logloss:0.22643
[347]   validation_0-logloss:0.22636
[348]   validation_0-logloss:0.22629
[349]   validation_0-logloss:0.22622
[350]   validation_0-logloss:0.22615
[351]   validation_0-logloss:0.22608
[352]   validation_0-logloss:0.22602
[353]   validation_0-logloss:0.22595
[354]   validation_0-logloss:0.22589
[355]   validation_0-logloss:0.22582
[356]   validation_0-logloss:0.22576
[357]   validation_0-logloss:0.22570
[358]   validation_0-logloss:0.22564
[359]   validation_0-logloss:0.22558
[360]   validation_0-logloss:0.22552
[361]   validation_0-logloss:0.22546
[362]   validation_0-logloss:0.22540
[363]   validation_0-logloss:0.22534
[364]   validation_0-logloss:0.22528
[365]   validation_0-logloss:0.22522
[366]   validation_0-logloss:0.22516
[367]   validation_0-logloss:0.22510
[368]   validation_0-logloss:0.22505
```

```
[369]   validation_0-logloss:0.22500
[370]   validation_0-logloss:0.22494
[371]   validation_0-logloss:0.22489
[372]   validation_0-logloss:0.22484
[373]   validation_0-logloss:0.22479
[374]   validation_0-logloss:0.22473
[375]   validation_0-logloss:0.22469
[376]   validation_0-logloss:0.22464
[377]   validation_0-logloss:0.22459
[378]   validation_0-logloss:0.22454
[379]   validation_0-logloss:0.22450
[380]   validation_0-logloss:0.22445
[381]   validation_0-logloss:0.22440
[382]   validation_0-logloss:0.22436
[383]   validation_0-logloss:0.22431
[384]   validation_0-logloss:0.22427
[385]   validation_0-logloss:0.22422
[386]   validation_0-logloss:0.22418
[387]   validation_0-logloss:0.22414
[388]   validation_0-logloss:0.22409
[389]   validation_0-logloss:0.22405
[390]   validation_0-logloss:0.22401
[391]   validation_0-logloss:0.22396
[392]   validation_0-logloss:0.22392
[393]   validation_0-logloss:0.22388
[394]   validation_0-logloss:0.22384
[395]   validation_0-logloss:0.22380
[396]   validation_0-logloss:0.22376
[397]   validation_0-logloss:0.22372
[398]   validation_0-logloss:0.22369
[399]   validation_0-logloss:0.22365
[400]   validation_0-logloss:0.22361
[401]   validation_0-logloss:0.22358
[402]   validation_0-logloss:0.22354
[403]   validation_0-logloss:0.22351
[404]   validation_0-logloss:0.22347
[405]   validation_0-logloss:0.22344
[406]   validation_0-logloss:0.22341
[407]   validation_0-logloss:0.22337
[408]   validation_0-logloss:0.22335
[409]   validation_0-logloss:0.22332
```

```
[410]   validation_0-logloss:0.22329
[411]   validation_0-logloss:0.22326
[412]   validation_0-logloss:0.22323
[413]   validation_0-logloss:0.22320
[414]   validation_0-logloss:0.22317
[415]   validation_0-logloss:0.22315
[416]   validation_0-logloss:0.22312
[417]   validation_0-logloss:0.22309
[418]   validation_0-logloss:0.22307
[419]   validation_0-logloss:0.22304
[420]   validation_0-logloss:0.22301
[421]   validation_0-logloss:0.22299
[422]   validation_0-logloss:0.22297
[423]   validation_0-logloss:0.22294
[424]   validation_0-logloss:0.22292
[425]   validation_0-logloss:0.22289
[426]   validation_0-logloss:0.22287
[427]   validation_0-logloss:0.22284
[428]   validation_0-logloss:0.22282
[429]   validation_0-logloss:0.22280
[430]   validation_0-logloss:0.22277
[431]   validation_0-logloss:0.22275
[432]   validation_0-logloss:0.22273
[433]   validation_0-logloss:0.22271
[434]   validation_0-logloss:0.22268
[435]   validation_0-logloss:0.22266
[436]   validation_0-logloss:0.22264
[437]   validation_0-logloss:0.22262
[438]   validation_0-logloss:0.22260
[439]   validation_0-logloss:0.22258
[440]   validation_0-logloss:0.22256
[441]   validation_0-logloss:0.22254
[442]   validation_0-logloss:0.22252
[443]   validation_0-logloss:0.22250
[444]   validation_0-logloss:0.22248
[445]   validation_0-logloss:0.22246
[446]   validation_0-logloss:0.22244
[447]   validation_0-logloss:0.22242
[448]   validation_0-logloss:0.22241
[449]   validation_0-logloss:0.22239
[450]   validation_0-logloss:0.22237
```

```
[451]   validation_0-logloss:0.22236
[452]   validation_0-logloss:0.22234
[453]   validation_0-logloss:0.22232
[454]   validation_0-logloss:0.22230
[455]   validation_0-logloss:0.22228
[456]   validation_0-logloss:0.22227
[457]   validation_0-logloss:0.22225
[458]   validation_0-logloss:0.22224
[459]   validation_0-logloss:0.22222
[460]   validation_0-logloss:0.22220
[461]   validation_0-logloss:0.22219
[462]   validation_0-logloss:0.22217
[463]   validation_0-logloss:0.22216
[464]   validation_0-logloss:0.22214
[465]   validation_0-logloss:0.22213
[466]   validation_0-logloss:0.22211
[467]   validation_0-logloss:0.22209
[468]   validation_0-logloss:0.22208
[469]   validation_0-logloss:0.22206
[470]   validation_0-logloss:0.22205
[471]   validation_0-logloss:0.22203
[472]   validation_0-logloss:0.22202
[473]   validation_0-logloss:0.22201
[474]   validation_0-logloss:0.22199
[475]   validation_0-logloss:0.22198
[476]   validation_0-logloss:0.22197
[477]   validation_0-logloss:0.22195
[478]   validation_0-logloss:0.22194
[479]   validation_0-logloss:0.22193
[480]   validation_0-logloss:0.22191
[481]   validation_0-logloss:0.22190
[482]   validation_0-logloss:0.22189
[483]   validation_0-logloss:0.22188
[484]   validation_0-logloss:0.22186
[485]   validation_0-logloss:0.22185
[486]   validation_0-logloss:0.22184
[487]   validation_0-logloss:0.22183
[488]   validation_0-logloss:0.22181
[489]   validation_0-logloss:0.22180
[490]   validation_0-logloss:0.22179
[491]   validation_0-logloss:0.22178
```

```
[492]   validation_0-logloss:0.22177
[493]   validation_0-logloss:0.22176
[494]   validation_0-logloss:0.22174
[495]   validation_0-logloss:0.22173
[496]   validation_0-logloss:0.22172
[497]   validation_0-logloss:0.22172
[498]   validation_0-logloss:0.22170
[499]   validation_0-logloss:0.22169
```

Out[47]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
```

In [48]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score

y_pred_prob = xgb_model.predict(X_val)
y_pred = (y_pred_prob > 0.5).astype(int)

print("Accuracy:", accuracy_score(y_val, y_pred))
print("Precision:", precision_score(y_val, y_pred))
print("Recall:", recall_score(y_val, y_pred))
print("F1:", f1_score(y_val, y_pred))
print("ROC AUC:", roc_auc_score(y_val, y_pred_prob))
```

```
Accuracy: 0.9101072532266861
Precision: 0.8950273016614594
Recall: 0.9291018762692855
F1: 0.91174633391832
ROC AUC: 0.9101158816554326
```

In [49]:
```
#random forest
from sklearn.ensemble import RandomForestClassifier
rfc_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=30,
    random_state=42
)



rfc_model.fit(X_train, y_train)
```

Out[49]:
```
RandomForestClassifier(max_depth=30, random_state=42)
```

In [50]:
```
y_pred_prob = rfc_model.predict(X_val)
y_pred = (y_pred_prob > 0.5).astype(int)

print("Accuracy:", accuracy_score(y_val, y_pred))
print("Precision:", precision_score(y_val, y_pred))
print("Recall:", recall_score(y_val, y_pred))
print("F1:", f1_score(y_val, y_pred))
print("ROC AUC:", roc_auc_score(y_val, y_pred_prob))
```

```
Accuracy: 0.9100315094225292
Precision: 0.8961921499707088
Recall: 0.9274044436361432
F1: 0.9115311852944243
ROC AUC: 0.910039401188775
```

In [51]:
```
#adding xgboost with cnn with handcrafted features
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
import numpy as np
import xgboost as xgb
import pandas as pd
```

In [53]:
```
import pandas as pd
import re
import math
from urllib.parse import urlparse
import tldextract
from rapidfuzz import process, fuzz



import pickle
f= open('tld_encoding_serise_dataset_1.bin','rb')
tld_stats = pickle.load(file=f)
f.close()
print(type(tld_stats))

import kagglehub
import os
# Download latest version
folder_path = kagglehub.dataset_download("cheedcheed/top1m")

print("Path to dataset files:", folder_path)
csv_files = [f for f in os.listdir(folder_path) if f.lower().endswith('.csv')]

if not csv_files:
    raise FileNotFoundError("No CSV files found in the folder!")

# read the first CSV file
file_path = os.path.join(folder_path, csv_files[0])
alexa_top_1m_domain = pd.read_csv(file_path,header=None,names=['rank', 'domain'])
alexa_domains_set = set(alexa_top_1m_domain['domain'].apply(str.lower))
```

```python
# --- Helper function: Shannon entropy ---
def safe_parse(url: str):
    """Safely parse URLs, adding http:// if missing and handling bad IPv6 parts."""
    if not isinstance(url, str) or not url.strip():
        return urlparse("http://")

    # Ensure scheme exists
    if not re.match(r'^[a-zA-Z]+://', url):
        url = 'http://' + url

    # Clean invalid brackets that trigger IPv6 errors
    url = re.sub(r'\[.*?\]', '', url)

    try:
        return urlparse(url)
    except ValueError:
        # fallback: strip more aggressively if still malformed
        url = re.sub(r'[^a-zA-Z0-9:/._\-?&=]', '', url)
        return urlparse(url)
def calculate_entropy(string):
    """Measures randomness of characters in the URL."""
    if not string:
        return 0
    freq = {char: string.count(char) for char in set(string)}
    entropy = -sum((count / len(string)) * math.log2(count / len(string)) for count in freq.values())
    return entropy

# --- Main feature extraction function ---
def extract_handcrafted_features(url):
    features = {}
    if not re.match(r'^[hH]+[tT]+[tT]+[pP]+[sS]+://', url):
        url = 'http://' + url
    parsed = safe_parse(url)

    # 1️⃣ Basic structural features
    features['url_length'] = len(url)
    features['hostname_length'] = len(parsed.netloc)
    features['path_length'] = len(parsed.path)
    features['num_dots'] = url.count('.')
    features['num_hyphens'] = url.count('-')
    features['num_digits'] = sum(c.isdigit() for c in url)
```

```
    features['num_letters'] = sum(c.isalpha() for c in url)
    features['num_params'] = url.count('?')
    features['num_equals'] = url.count('=')
    features['num_slashes'] = url.count('/')
    features['num_at'] = url.count('@')

    # 2️⃣ Lexical / composition cues
    features['has_https'] = 1 if url.lower().startswith('https') else 0
    features['has_ip'] = 1 if re.search(r'(\d{1,3}\.){3}\d{1,3}', parsed.netloc) else 0
    features['has_subdomain'] = 1 if parsed.netloc.count('.') > 1 else 0
    features['has_suspicious_words'] = 1 if re.search(r'(login|secure|verify|update|free|bank|click)', url.lower()) else 0

    # 3️⃣ Domain / TLD features
    extracted = tldextract.extract(url)
    main_domain = f"{extracted.domain}.{extracted.suffix}"
    if ':' in main_domain:  # remove port
        main_domain = main_domain.split(':')[0]
    features['domain_length'] = len(main_domain)
    features['in_alexa_top1m'] = 1 if main_domain in alexa_domains_set else 0
    '''
    if features['in_alexa_top1m'] == 0 and main_domain:  # only check if domain not in top1M
        # find closest match in Alexa domains
        best_match, score, _ = process.extractOne(main_domain, alexa_domains_set, scorer=fuzz.ratio)
        features['closest_alexa_domain'] = best_match
        features['closest_alexa_score'] = score  # 0-100
    else:
        features['closest_alexa_score'] = 1000  # high score to show that it is original url
    '''
    ext = tldextract.extract(url)
    tld = ext.suffix    # "com", "co.uk", "org"
    features['tld'] = tld if tld else 'unknown'
    features['tld_phish_ratio'] = tld_stats['phish_ratio'].get(features['tld'], 0.5)
    features['tld_total_frequency'] = tld_stats['total'].get(features['tld'], 1)

    # 4️⃣ Ratios
    features['digit_ratio'] = features['num_digits'] / (features['url_length'] + 1e-5)
    features['special_char_ratio'] = (features['num_hyphens'] + features['num_dots'] + features['num_slashes']) / (features['url_

    # 5️⃣ Entropy (measures randomness / obfuscation)
    features['url_entropy'] = calculate_entropy(url)
```

```
    # 6️⃣ Misplacement indicators
    # '@' symbol used to hide real domain (like "http://evil.com@legit.com")
    features['at_in_domain'] = 1 if '@' in parsed.netloc else 0

    # Double slashes '//' appearing after path (used to trick users)
    features['double_slash_in_path'] = 1 if re.search(r'/.+//', parsed.path) else 0

    return features
```

```
<class 'pandas.core.frame.DataFrame'>
Path to dataset files: C:\Users\rrpra\.cache\kagglehub\datasets\cheedcheed\top1m\versions\1
```

In [54]:
```
from tqdm import tqdm
tqdm.pandas()
for i in encoded_data:

    for split in encoded_data[i]:
        print("feature extracting", i, split)
        encoded_data[i][split] = encoded_data[i][split].assign(**encoded_data[i][split].url.progress_apply(lambda url : pd.Series
```

```
feature extracting Dataset 1 (Malicious URLs) train
100%|██████████| 417732/417732 [01:27<00:00, 4794.61it/s]
feature extracting Dataset 1 (Malicious URLs) valid
100%|██████████| 52217/52217 [00:10<00:00, 4781.51it/s]
feature extracting Dataset 1 (Malicious URLs) test
100%|██████████| 52217/52217 [00:10<00:00, 5047.86it/s]
feature extracting Dataset 2 (ndarvind/phiusiil-phishing) train
100%|██████████| 188296/188296 [00:37<00:00, 5080.37it/s]
feature extracting Dataset 2 (ndarvind/phiusiil-phishing) valid
100%|██████████| 23537/23537 [00:04<00:00, 5824.99it/s]
feature extracting Dataset 2 (ndarvind/phiusiil-phishing) test
100%|██████████| 23537/23537 [00:03<00:00, 5925.99it/s]
feature extracting Dataset 3 (kmack/Phishing_urls) train
100%|██████████| 528097/528097 [01:44<00:00, 5077.27it/s]
feature extracting Dataset 3 (kmack/Phishing_urls) valid
100%|██████████| 66012/66012 [00:12<00:00, 5118.95it/s]
feature extracting Dataset 3 (kmack/Phishing_urls) test
100%|██████████| 66013/66013 [00:13<00:00, 5028.54it/s]
```

In [55]:
```
# Split features and labels
independent_features = ['url_length', 'hostname_length', 'path_length',
    'num_dots', 'num_hyphens', 'num_digits', 'num_letters', 'num_params',
    'num_equals', 'num_slashes', 'num_at', 'has_https', 'has_ip',
    'has_subdomain', 'has_suspicious_words', 'domain_length',
    'in_alexa_top1m', 'tld_phish_ratio', 'tld_total_frequency',
    'digit_ratio', 'special_char_ratio', 'url_entropy', 'at_in_domain',
    'double_slash_in_path']
dependet_features  = 'label'
model_dict = {}
for i in encoded_data:
    print(f"train for {i}")
    X_train = encoded_data[i]['train'][independent_features]
    y_train = encoded_data[i]['train'][dependet_features]
    X_test = encoded_data[i]['valid'][independent_features]
    y_test = encoded_data[i]['valid'][dependet_features]

    # Create XGBoost classifier
    model_dict[i] = xgb.XGBClassifier(
        n_estimators=100,      # number of boosting rounds
        learning_rate=0.01,     # step size shrinkage
        max_depth=30,           # tree depth
        eval_metric='logloss', # evaluation metric
        use_label_encoder=False
    )

    # Train the model
    model_dict[i].fit(X_train, y_train)
    # Predict
    y_pred = model_dict[i] .predict(X_test)

    # Evaluate
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
train for Dataset 1 (Malicious URLs)
```

```
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\training.py:199: UserWarning: [07:48:56] WARNING:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Accuracy: 0.9650496964590076

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     42808
           1       0.97      0.83      0.90      9409

    accuracy                           0.97     52217
   macro avg       0.97      0.91      0.94     52217
weighted avg       0.97      0.97      0.96     52217

train for Dataset 2 (ndarvind/phiusiil-phishing)
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\training.py:199: UserWarning: [07:49:08] WARNING:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Accuracy: 0.9973233632153631

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     10052
           1       1.00      1.00      1.00     13485

    accuracy                           1.00     23537
   macro avg       1.00      1.00      1.00     23537
weighted avg       1.00      1.00      1.00     23537

train for Dataset 3 (kmack/Phishing_urls)
```

```
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\training.py:199: UserWarning: [07:49:09] WARNING:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Accuracy: 0.8882778888686905

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.87      0.89     33021
           1       0.88      0.90      0.89     32991

    accuracy                           0.89     66012
   macro avg       0.89      0.89      0.89     66012
weighted avg       0.89      0.89      0.89     66012
```

In [56]:
```
#logistic_regression(nn+xgboost)
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import StratifiedKFold
independent_features = ['url_length', 'hostname_length', 'path_length',
    'num_dots', 'num_hyphens', 'num_digits', 'num_letters', 'num_params',
    'num_equals', 'num_slashes', 'num_at', 'has_https', 'has_ip',
    'has_subdomain', 'has_suspicious_words', 'domain_length',
    'in_alexa_top1m', 'tld_phish_ratio', 'tld_total_frequency',
    'digit_ratio', 'special_char_ratio', 'url_entropy', 'at_in_domain',
    'double_slash_in_path']
dependent_features  = 'label'
logistic_model = {}
for name in encoded_data:
    #print(model_dict[name])
    #print(nn_model[name])
    nn_model[name].eval()
    with torch.no_grad():
        model = nn_model[name]
        #print(model)
        x_ = encoded_data[name]['train']['encode']
        x_np = np.stack(x_.values)
```

```
        x_ = torch.tensor(x_np, dtype=torch.long).to(device)  # move once to GPU/CPU where model is
        batch_size = 512  # adjust for your GPU memory
        nn_preds_train = []

        #  🔹 Process manually in batches
        for i in tqdm(range(0, len(x_), batch_size), desc=f"Predicting NN for {name}", ncols=80):
            batch_x = x_[i:i + batch_size]
            outputs = model(batch_x)
            nn_preds_train.append(outputs)

    #  🔹 Combine all predictions
    nn_preds_train = torch.cat(nn_preds_train, dim=0).cpu()

    # XGBoost predictions
    xgb_preds_train = model_dict[name].predict_proba(encoded_data[name]['train'][independent_features])[:, 1]

    # Stack predictions as new features
    meta_X = np.column_stack((nn_preds_train, xgb_preds_train))
    meta_y = encoded_data[name]['train'][dependent_features]


    meta_model= LogisticRegressionCV(
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    penalty="l2",
    scoring="roc_auc",
    solver="lbfgs",
    Cs=10,
    max_iter=1000,
    n_jobs=-1
)
    meta_model.fit(meta_X, meta_y)


    with torch.no_grad():
        x_ = encoded_data[name]['valid']['encode']
        x_np = np.stack(x_.values)
        x_ = torch.tensor(x_np, dtype=torch.long).to(device)  # move once to GPU/CPU where model is
        batch_size = 512  # adjust for your GPU memory
        nn_preds_val = []

        #  🔹 Process manually in batches
```

```
        for i in tqdm(range(0, len(x_), batch_size), desc=f"Predicting NN for {name}", ncols=80):
            batch_x = x_[i:i + batch_size]
            outputs = model(batch_x)
            nn_preds_val.append(outputs)

    #  🔹 Combine all predictions
    nn_preds_val = torch.cat(nn_preds_val, dim=0).cpu()
    # XGBoost predictions
    xgb_preds_val = model_dict[name].predict_proba(encoded_data[name]['valid'][independent_features])[:, 1]

    # Stack predictions as new features
    meta_X_val = np.column_stack((nn_preds_val, xgb_preds_val))
    meta_y_val = encoded_data[name]['valid'][dependent_features]


    y_pred_train = meta_model.predict(meta_X)
    y_pred_prob_train = meta_model.predict_proba(meta_X)[:, 1]

    print("\n📘 TRAIN METRICS")
    print("Accuracy :", accuracy_score(meta_y, y_pred_train))
    print("Precision:", precision_score(meta_y, y_pred_train))
    print("Recall   :", recall_score(meta_y, y_pred_train))
    print("F1-score :", f1_score(meta_y, y_pred_train))
    print("ROC AUC  :", roc_auc_score(meta_y, y_pred_prob_train))

    # ==============================
    #  🔹 Validation Metrics
    # ==============================
    y_pred_val = meta_model.predict(meta_X_val)
    y_pred_prob_val = meta_model.predict_proba(meta_X_val)[:, 1]

    print("\n📗 VALIDATION METRICS")
    print("Accuracy :", accuracy_score(meta_y_val, y_pred_val))
    print("Precision:", precision_score(meta_y_val, y_pred_val))
    print("Recall   :", recall_score(meta_y_val, y_pred_val))
    print("F1-score :", f1_score(meta_y_val, y_pred_val))
    print("ROC AUC  :", roc_auc_score(meta_y_val, y_pred_prob_val))
```

```
Predicting NN for Dataset 1 (Malicious URLs): 100%|██| 816/816 [03:29<00:00,  3.8
Predicting NN for Dataset 1 (Malicious URLs): 100%|██| 102/102 [00:25<00:00,  3.9
```

```
📘 TRAIN METRICS
Accuracy : 0.9858354160083499
Precision: 0.9736640804283685
Recall   : 0.9470034144202792
F1-score : 0.9601487099011968
ROC AUC  : 0.9971999085276664

📗 VALIDATION METRICS
Accuracy : 0.9802363215044909
Precision: 0.9608317746726812
Recall   : 0.928153895206717
F1-score : 0.9442101848848524
ROC AUC  : 0.9945813634182346
Predicting NN for Dataset 2 (ndarvind/phiusiil-phishing): 100%|██| 368/368 [01:32
Predicting NN for Dataset 2 (ndarvind/phiusiil-phishing): 100%|██| 46/46 [00:11<0
📘 TRAIN METRICS
Accuracy : 0.998284615711433
Precision: 0.9970791854994501
Recall   : 0.9999351130886169
F1-score : 0.9985051071648409
ROC AUC  : 0.9991190365969227

📗 VALIDATION METRICS
Accuracy : 0.9978331987933892
Precision: 0.9963789535914869
Recall   : 0.999851687059696
F1-score : 0.998112299663175
ROC AUC  : 0.9991543786916857
Predicting NN for Dataset 3 (kmack/Phishing_urls): 100%|██| 1032/1032 [04:15<00:0
Predicting NN for Dataset 3 (kmack/Phishing_urls): 100%|██| 129/129 [00:31<00:00,
```

```
📘 TRAIN METRICS
Accuracy : 0.9402477196424142
Precision: 0.9342073300620742
Recall   : 0.9471446758206784
F1-score : 0.9406315203260139
ROC AUC  : 0.9876472303850408

📗 VALIDATION METRICS
Accuracy : 0.8996849057747076
Precision: 0.8908950754543568
Recall   : 0.9108241641659847
F1-score : 0.9007494004796163
ROC AUC  : 0.9671546864429791
```

In [57]:

```python
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
independent_features = ['url_length', 'hostname_length', 'path_length',
    'num_dots', 'num_hyphens', 'num_digits', 'num_letters', 'num_params',
    'num_equals', 'num_slashes', 'num_at', 'has_https', 'has_ip',
    'has_subdomain', 'has_suspicious_words', 'domain_length',
    'in_alexa_top1m', 'tld_phish_ratio', 'tld_total_frequency',
    'digit_ratio', 'special_char_ratio', 'url_entropy', 'at_in_domain',
    'double_slash_in_path']
dependent_features  = 'label'
for name in encoded_data:
    #print(model_dict[name])
    #print(nn_model[name])
    nn_model[name].eval()
    with torch.no_grad():
        model = nn_model[name]
        #print(model)
        x_ = encoded_data[name]['train']['encode']
        x_np = np.stack(x_.values)
        x_ = torch.tensor(x_np, dtype=torch.long).to(device)  # move once to GPU/CPU where model is
        batch_size = 512  # adjust for your GPU memory
        nn_preds_train = []

        #  🔹 Process manually in batches
        for i in tqdm(range(0, len(x_), batch_size), desc=f"Predicting NN for {name}", ncols=80):
            batch_x = x_[i:i + batch_size]
            outputs = model(batch_x)
```

```
        nn_preds_train.append(outputs)

# 🔹 Combine all predictions
nn_preds_train = torch.cat(nn_preds_train, dim=0).cpu()

# XGBoost predictions
xgb_preds_train = model_dict[name].predict_proba(encoded_data[name]['train'][independent_features])[:, 1]

# Stack predictions as new features
meta_X = np.column_stack((nn_preds_train, xgb_preds_train))
meta_y = encoded_data[name]['train'][dependent_features]


params = {
"hidden_layer_sizes": [(8,), (16,), (24,)],
"alpha": [1e-3, 1e-4, 1e-5],  # L2 regularization
"learning_rate_init": [1e-3],
"early_stopping": [True],
"max_iter": [500],
}

mlp = MLPClassifier(random_state=42)
meta_model = GridSearchCV(mlp, params, cv=StratifiedKFold(3), scoring="accuracy", n_jobs=-1, verbose=3)
meta_model.fit(meta_X, meta_y)


with torch.no_grad():
    x_ = encoded_data[name]['valid']['encode']
    x_np = np.stack(x_.values)
    x_ = torch.tensor(x_np, dtype=torch.long).to(device)  # move once to GPU/CPU where model is
    batch_size = 512  # adjust for your GPU memory
    nn_preds_val = []

    # 🔹 Process manually in batches
    for i in tqdm(range(0, len(x_), batch_size), desc=f"Predicting NN for {name}", ncols=80):
        batch_x = x_[i:i + batch_size]
        outputs = model(batch_x)
        nn_preds_val.append(outputs)

# 🔹 Combine all predictions
nn_preds_val = torch.cat(nn_preds_val, dim=0).cpu()
```

```
    # XGBoost predictions
    xgb_preds_val = model_dict[name].predict_proba(encoded_data[name]['valid'][independent_features])[:, 1]

    # Stack predictions as new features
    meta_X_val = np.column_stack((nn_preds_val, xgb_preds_val))
    meta_y_val = encoded_data[name]['valid'][dependent_features]


    y_pred_train = meta_model.predict(meta_X)
    y_pred_prob_train = meta_model.predict_proba(meta_X)[:, 1]

    print("\n📘 TRAIN METRICS")
    print("Accuracy :", accuracy_score(meta_y, y_pred_train))
    print("Precision:", precision_score(meta_y, y_pred_train))
    print("Recall   :", recall_score(meta_y, y_pred_train))
    print("F1-score :", f1_score(meta_y, y_pred_train))
    print("ROC AUC  :", roc_auc_score(meta_y, y_pred_prob_train))

    # ==============================
    #  🔹 Validation Metrics
    # ==============================
    y_pred_val = meta_model.predict(meta_X_val)
    y_pred_prob_val = meta_model.predict_proba(meta_X_val)[:, 1]

    print("\n📗 VALIDATION METRICS")
    print("Accuracy :", accuracy_score(meta_y_val, y_pred_val))
    print("Precision:", precision_score(meta_y_val, y_pred_val))
    print("Recall   :", recall_score(meta_y_val, y_pred_val))
    print("F1-score :", f1_score(meta_y_val, y_pred_val))
    print("ROC AUC  :", roc_auc_score(meta_y_val, y_pred_prob_val))
```

```
Predicting NN for Dataset 1 (Malicious URLs): 100%|██| 816/816 [03:33<00:00,  3.8
Fitting 3 folds for each of 9 candidates, totalling 27 fits
Predicting NN for Dataset 1 (Malicious URLs): 100%|██| 102/102 [00:25<00:00,  3.9
```

```
📘 TRAIN METRICS
Accuracy : 0.986010169199391
Precision: 0.97210548505991
Recall   : 0.9496074080962946
F1-score : 0.9607247506653405
ROC AUC  : 0.997221248084375

📗 VALIDATION METRICS
Accuracy : 0.9798533044793841
Precision: 0.9587221429355582
Recall   : 0.928153895206717
F1-score : 0.9431904093314613
ROC AUC  : 0.9945915612810543
Predicting NN for Dataset 2 (ndarvind/phiusiil-phishing): 100%|██| 368/368 [01:40
Fitting 3 folds for each of 9 candidates, totalling 27 fits
Predicting NN for Dataset 2 (ndarvind/phiusiil-phishing): 100%|██| 46/46 [00:12<0
📘 TRAIN METRICS
Accuracy : 0.998327102009602
Precision: 0.9971713025874261
Recall   : 0.999916573971079
F1-score : 0.9985420514031019
ROC AUC  : 0.9991221536151611

📗 VALIDATION METRICS
Accuracy : 0.997875685091558
Precision: 0.9964525903480895
Recall   : 0.999851687059696
F1-score : 0.9981492448919159
ROC AUC  : 0.999151977385375
Predicting NN for Dataset 3 (kmack/Phishing_urls): 100%|██| 1032/1032 [04:57<00:0
Fitting 3 folds for each of 9 candidates, totalling 27 fits
Predicting NN for Dataset 3 (kmack/Phishing_urls): 100%|██| 129/129 [00:35<00:00,
```

```
📘 TRAIN METRICS
Accuracy : 0.942730218122807
Precision: 0.9436656085298987
Recall   : 0.9416204419387105
F1-score : 0.9426419159314525
ROC AUC  : 0.9876519151601969

📗 VALIDATION METRICS
Accuracy : 0.8959431618493607
Precision: 0.8941396583982135
Recall   : 0.8981237307144373
F1-score : 0.8961272664035446
ROC AUC  : 0.9660475388040572
```

In [58]:
```python
#xgboost(nn+xgboost)
from sklearn.linear_model import LogisticRegression

independent_features = ['url_length', 'hostname_length', 'path_length',
    'num_dots', 'num_hyphens', 'num_digits', 'num_letters', 'num_params',
    'num_equals', 'num_slashes', 'num_at', 'has_https', 'has_ip',
    'has_subdomain', 'has_suspicious_words', 'domain_length',
    'in_alexa_top1m', 'tld_phish_ratio', 'tld_total_frequency',
    'digit_ratio', 'special_char_ratio', 'url_entropy', 'at_in_domain',
    'double_slash_in_path']
dependent_features  = 'label'
logistic_model = {}
for name in encoded_data:
    #print(model_dict[name])
    #print(nn_model[name])
    nn_model[name].eval()
    with torch.no_grad():
        model = nn_model[name]
        #print(model)
        x_ = encoded_data[name]['train']['encode']
        x_np = np.stack(x_.values)
        x_ = torch.tensor(x_np, dtype=torch.long).to(device)  # move once to GPU/CPU where model is
        batch_size = 512  # adjust for your GPU memory
        nn_preds_train = []

        #  🔹 Process manually in batches
        for i in tqdm(range(0, len(x_), batch_size), desc=f"Predicting NN for {name}", ncols=80):
```

```
        batch_x = x_[i:i + batch_size]
        outputs = model(batch_x)
        nn_preds_train.append(outputs)

#  🔹 Combine all predictions
nn_preds_train = torch.cat(nn_preds_train, dim=0).cpu()

# XGBoost predictions
xgb_preds_train = model_dict[name].predict_proba(encoded_data[name]['train'][independent_features])[:, 1]

# Stack predictions as new features
meta_X = np.column_stack((nn_preds_train, xgb_preds_train))
meta_y = encoded_data[name]['train'][dependent_features]

with torch.no_grad():
    x_ = encoded_data[name]['valid']['encode']
    x_np = np.stack(x_.values)
    x_ = torch.tensor(x_np, dtype=torch.long).to(device)  # move once to GPU/CPU where model is
    batch_size = 512  # adjust for your GPU memory
    nn_preds_val = []

    #  🔹 Process manually in batches
    for i in tqdm(range(0, len(x_), batch_size), desc=f"Predicting NN for {name}", ncols=80):
        batch_x = x_[i:i + batch_size]
        outputs = model(batch_x)
        nn_preds_val.append(outputs)

#  🔹 Combine all predictions
nn_preds_val = torch.cat(nn_preds_val, dim=0).cpu()
# XGBoost predictions
xgb_preds_val = model_dict[name].predict_proba(encoded_data[name]['valid'][independent_features])[:, 1]

# Stack predictions as new features
meta_X_val = np.column_stack((nn_preds_val, xgb_preds_val))
meta_y_val = encoded_data[name]['valid'][dependent_features]


xgb_model = xgb.XGBClassifier(
    n_estimators=500,
    learning_rate=0.01,
    bjective='binary:logistic',
```

```
    eval_metric='logloss',
    use_label_encoder=False,
    max_depth = 10,
    random_state=42,

)

xgb_model.fit(
    meta_X, meta_y,
    #eval_set=[(meta_X_val, meta_y_val)],
)


y_pred_train = meta_model.predict(meta_X)
y_pred_prob_train = meta_model.predict_proba(meta_X)[:, 1]

print("\n📘 TRAIN METRICS")
print("Accuracy :", accuracy_score(meta_y, y_pred_train))
print("Precision:", precision_score(meta_y, y_pred_train))
print("Recall   :", recall_score(meta_y, y_pred_train))
print("F1-score :", f1_score(meta_y, y_pred_train))
print("ROC AUC  :", roc_auc_score(meta_y, y_pred_prob_train))

# ==============================
#  🔹 Validation Metrics
# ==============================
y_pred_val = meta_model.predict(meta_X_val)
y_pred_prob_val = meta_model.predict_proba(meta_X_val)[:, 1]

print("\n📗 VALIDATION METRICS")
print("Accuracy :", accuracy_score(meta_y_val, y_pred_val))
print("Precision:", precision_score(meta_y_val, y_pred_val))
print("Recall   :", recall_score(meta_y_val, y_pred_val))
print("F1-score :", f1_score(meta_y_val, y_pred_val))
print("ROC AUC  :", roc_auc_score(meta_y_val, y_pred_prob_val))
```

```
Predicting NN for Dataset 1 (Malicious URLs): 100%|█| 816/816 [03:42<00:00,  3.6
Predicting NN for Dataset 1 (Malicious URLs): 100%|█| 102/102 [00:28<00:00,  3.6
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\training.py:199: UserWarning: [08:18:25] WARNING:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790:
Parameters: { "bjective", "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

📘 TRAIN METRICS
Accuracy : 0.9785819616404776
Precision: 0.9888839746424886
Recall   : 0.8911504072061539
F1-score : 0.9374768516900887
ROC AUC : 0.9968031453477875

📗 VALIDATION METRICS
Accuracy : 0.9693394871402033
Precision: 0.9812623274161736
Recall   : 0.8459985120629184
F1-score : 0.9086239369898979
ROC AUC : 0.9935940973325043

```
Predicting NN for Dataset 2 (ndarvind/phiusiil-phishing): 100%|█| 368/368 [01:40
Predicting NN for Dataset 2 (ndarvind/phiusiil-phishing): 100%|█| 46/46 [00:12<0
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\training.py:199: UserWarning: [08:20:22] WARNING:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790:
Parameters: { "bjective", "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

📘 TRAIN METRICS
Accuracy : 0.9978650635170158
Precision: 0.9963149728471683
Recall   : 0.999972191323693
F1-score : 0.9981402320546272
ROC AUC : 0.9991111823818333

📗 VALIDATION METRICS
Accuracy : 0.9974933084080384
Precision: 0.9957902511078287
Recall   : 0.999851687059696
F1-score : 0.9978168362627197
ROC AUC : 0.9991954959903717

```
Predicting NN for Dataset 3 (kmack/Phishing_urls): 100%|██| 1032/1032 [04:42<00:0
Predicting NN for Dataset 3 (kmack/Phishing_urls): 100%|██| 129/129 [00:35<00:00,
c:\Users\rrpra\AppData\Local\Programs\Python\Python313\Lib\site-packages\xgboost\training.py:199: UserWarning: [08:25:47] WARNING:
C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:790:
Parameters: { "bjective", "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

📘 TRAIN METRICS
Accuracy : 0.942730218122807
Precision: 0.9436656085298987
Recall : 0.9416204419387105
F1-score : 0.9426419159314525
ROC AUC : 0.9876519151601969

📗 VALIDATION METRICS
Accuracy : 0.8959431618493607
Precision: 0.8941396583982135
Recall : 0.8981237307144373
F1-score : 0.8961272664035446
ROC AUC : 0.9660475388040572

==========

🚀 Training model on DATASET_2 dataset

_____🧩 Using 10% of training data_____

Epoch 1/6 | Train Loss: 0.6624, Train Acc: 0.6061 | Val Loss: 0.5799, Val Acc: 0.7219

Epoch 2/6 | Train Loss: 0.5179, Train Acc: 0.7594 | Val Loss: 0.5534, Val Acc: 0.7312

Epoch 3/6 | Train Loss: 0.4341, Train Acc: 0.8096 | Val Loss: 0.4320, Val Acc: 0.8120

Epoch 4/6 | Train Loss: 0.3945, Train Acc: 0.8308 | Val Loss: 0.4003, Val Acc: 0.8252

Epoch 5/6 | Train Loss: 0.3716, Train Acc: 0.8404 | Val Loss: 0.3567, Val Acc: 0.8496

Epoch 6/6 | Train Loss: 0.3588, Train Acc: 0.8458 | Val Loss: 0.3478, Val Acc: 0.8523

_____🧩 Using 50% of training data_____

Epoch 1/6 | Train Loss: 0.5106, Train Acc: 0.7353 | Val Loss: 0.5287, Val Acc: 0.6781

Epoch 2/6 | Train Loss: 0.3578, Train Acc: 0.8452 | Val Loss: 0.5693, Val Acc: 0.6529

Epoch 3/6 | Train Loss: 0.3324, Train Acc: 0.8569 | Val Loss: 0.5073, Val Acc: 0.7133
Epoch 4/6 | Train Loss: 0.3191, Train Acc: 0.8620 | Val Loss: 0.5989, Val Acc: 0.6296
Epoch 5/6 | Train Loss: 0.3086, Train Acc: 0.8663 | Val Loss: 0.4869, Val Acc: 0.7649
Epoch 6/6 | Train Loss: 0.3027, Train Acc: 0.8693 | Val Loss: 0.4661, Val Acc: 0.7953
_____ 🧩 Using 100% of training data_____
Epoch 1/6 | Train Loss: 0.4213, Train Acc: 0.8050 | Val Loss: 0.3473, Val Acc: 0.8572
Epoch 2/6 | Train Loss: 0.3203, Train Acc: 0.8626 | Val Loss: 0.3535, Val Acc: 0.8393
Epoch 3/6 | Train Loss: 0.3016, Train Acc: 0.8692 | Val Loss: 0.4240, Val Acc: 0.7920
Epoch 4/6 | Train Loss: 0.2923, Train Acc: 0.8723 | Val Loss: 0.3619, Val Acc: 0.8312
Epoch 5/6 | Train Loss: 0.2845, Train Acc: 0.8753 | Val Loss: 0.3901, Val Acc: 0.8187
Epoch 6/6 | Train Loss: 0.2788, Train Acc: 0.8772 | Val Loss: 0.3979, Val Acc: 0.8223

✅ All datasets trained successfully!

==========

📈 Final Validation Accuracy Summary

dataset_2 | Val Acc: 0.8223 | Val Loss: 0.3979

In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import torch
```

In [ ]:
```
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for i in range(0, len(test_url_tensor), 64):  # batch size 64
        batch_x = test_url_tensor[i:i+64].to(device)
        batch_y = test_labels_tensor[i:i+64].to(device)
```

```
        outputs = model(batch_x)
        preds = (outputs >= 0.5).long().squeeze(1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch_y.cpu().numpy())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 6
      3 all_labels = []
      5 with torch.no_grad():
----> 6     for i in range(0, len(test_url_tensor), 64):  # batch size 64
      7         batch_x = test_url_tensor[i:i+64].to(device)
      8         batch_y = test_labels_tensor[i:i+64].to(device)

NameError: name 'test_url_tensor' is not defined
```

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F

class CNN_BiLSTM(nn.Module):
    def __init__(self, vocab_size, embed_dim=128):
        super(CNN_BiLSTM, self).__init__()

        # Embedding layer
        self.embedding = nn.Embedding(num_embeddings=vocab_size, embedding_dim=embed_dim)

        # Conv blocks
        self.conv1 = nn.Conv1d(embed_dim, 256, kernel_size=8, padding=4)
        self.pool1 = nn.MaxPool1d(2)

        self.conv2 = nn.Conv1d(256, 128, kernel_size=5, padding=2)
        self.pool2 = nn.MaxPool1d(2)

        # BiLSTM
        self.lstm = nn.LSTM(input_size=128, hidden_size=64, num_layers=1,
                            batch_first=True, bidirectional=True)

        # Fully connected layers
```

```
        self.fc1 = nn.Linear(64*2, 128)  # 64*2 because bidirectional
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, 1)  # binary output

    def forward(self, x):
        # x: (batch_size, seq_len)
        x = self.embedding(x)  # (batch_size, seq_len, embed_dim)
        x = x.permute(0, 2, 1)  # (batch_size, embed_dim, seq_len) for Conv1d

        # Conv block 1
        x = F.relu(self.conv1(x))
        x = self.pool1(x)

        # Conv block 2
        x = F.relu(self.conv2(x))
        x = self.pool2(x)

        # Prepare for LSTM: (batch_size, seq_len, features)
        x = x.permute(0, 2, 1)

        # BiLSTM
        lstm_out, _ = self.lstm(x)  # lstm_out: (batch_size, seq_len, 2*hidden_size)

        # Take the last timestep
        x = lstm_out[:, -1, :]  # (batch_size, 128)

        # Fully connected layers
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = self.fc2(x)

        return x
```

In [ ]:

```
import sys
import torch
import torch.nn as nn
import torch.nn.functional as F
from tqdm import tqdm
import matplotlib.pyplot as plt
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ============================================================
# 🔧 Training Config
# ============================================================
num_epochs = 3
lr = 0.001
batch_print_interval = 1

# Dictionary to store all dataset results
all_results = {}

# ============================================================
# 🔁 Loop Over Each Dataset
# ============================================================
for dataset_name, loaders in dataloader_dict.items():
    print("\n" + "="*70)
    print(f"🚀 Training CNN-BiLSTM on {dataset_name.upper()} dataset")
    print("="*70)

    train_loader = loaders["train_loader"]
    val_loader = loaders["val_loader"]

    model = CNN_BiLSTM(vocab_size=len(vocab)).to(device)
    criterion = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    # Track metrics
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        correct_train = 0
        total_train = 0

        for batch_idx, (batch_x, batch_y) in enumerate(train_loader):
            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device).float().unsqueeze(1)
            batch_y = torch.clamp(batch_y, 0, 1)
```

```
        optimizer.zero_grad()
        outputs = model(batch_x)
        loss = criterion(outputs, batch_y)
        loss.backward()
        optimizer.step()

        preds = (torch.sigmoid(outputs) >= 0.5).float()
        acc = (preds == batch_y).float().mean().item()

        train_loss += loss.item() * batch_x.size(0)
        correct_train += (preds == batch_y).sum().item()
        total_train += batch_x.size(0)

        if (batch_idx + 1) % batch_print_interval == 0:
            sys.stdout.write(f"\r[{dataset_name}] Epoch {epoch+1}/{num_epochs} | "
                             f"Batch {batch_idx+1}/{len(train_loader)} | "
                             f"Loss: {loss.item():.4f}, Acc: {acc:.4f}")
            sys.stdout.flush()

    avg_train_loss = train_loss / total_train
    train_acc = correct_train / total_train
    print()  # newline

    # === Validation ===
    model.eval()
    val_loss = 0
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for batch_x, batch_y in val_loader:
            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device).float().unsqueeze(1)
            batch_y = torch.clamp(batch_y, 0, 1)

            outputs = model(batch_x)
            loss = criterion(outputs, batch_y)
            val_loss += loss.item() * batch_x.size(0)

            preds = (torch.sigmoid(outputs) >= 0.5).float()
            correct_val += (preds == batch_y).sum().item()
```

```
                total_val += batch_x.size(0)

        avg_val_loss = val_loss / total_val
        val_acc = correct_val / total_val

        train_losses.append(avg_train_loss)
        val_losses.append(avg_val_loss)
        train_accs.append(train_acc)
        val_accs.append(val_acc)

        print(f"[{dataset_name}] Epoch {epoch+1}/{num_epochs} | "
              f"Train Loss: {avg_train_loss:.4f}, Train Acc: {train_acc:.4f} | "
              f"Val Loss: {avg_val_loss:.4f}, Val Acc: {val_acc:.4f}")

    # === Save results for this dataset ===
    all_results[dataset_name] = {
        "train_losses": train_losses,
        "val_losses": val_losses,
        "train_accs": train_accs,
        "val_accs": val_accs,
        "final_val_acc": val_accs[-1],
        "final_val_loss": val_losses[-1]
    }

print("\n✅ All datasets trained successfully!")

# ============================================================
# 📊 Summary of Final Results
# ============================================================
print("\n" + "="*70)
print("📈 Final Validation Summary (CNN-BiLSTM)")
print("="*70)
for name, res in all_results.items():
    print(f"{name:<20} | Val Acc: {res['final_val_acc']:.4f} | Val Loss: {res['final_val_loss']:.4f}")
```

```
======================================================================
🚀 Training CNN-BiLSTM on DATASET1 dataset
======================================================================
[dataset1] Epoch 1/3 | Batch 102/102 | Loss: 0.2697, Acc: 0.8942
[dataset1] Epoch 1/3 | Train Loss: 0.4063, Train Acc: 0.8384 | Val Loss: 0.2719, Val Acc: 0.8974
[dataset1] Epoch 2/3 | Batch 102/102 | Loss: 0.1222, Acc: 0.9658
[dataset1] Epoch 2/3 | Train Loss: 0.1782, Train Acc: 0.9378 | Val Loss: 0.1227, Val Acc: 0.9609
[dataset1] Epoch 3/3 | Batch 102/102 | Loss: 0.0869, Acc: 0.9730
[dataset1] Epoch 3/3 | Train Loss: 0.1033, Train Acc: 0.9677 | Val Loss: 0.0860, Val Acc: 0.9731

======================================================================
🚀 Training CNN-BiLSTM on DATASET2 dataset
======================================================================
[dataset2] Epoch 1/3 | Batch 47/47 | Loss: 0.0809, Acc: 0.9864
[dataset2] Epoch 1/3 | Train Loss: 0.3467, Train Acc: 0.8455 | Val Loss: 0.0146, Val Acc: 0.9977
[dataset2] Epoch 2/3 | Batch 47/47 | Loss: 0.0022, Acc: 1.0000
[dataset2] Epoch 2/3 | Train Loss: 0.0140, Train Acc: 0.9977 | Val Loss: 0.0102, Val Acc: 0.9981
[dataset2] Epoch 3/3 | Batch 47/47 | Loss: 0.0022, Acc: 1.0000
[dataset2] Epoch 3/3 | Train Loss: 0.0122, Train Acc: 0.9980 | Val Loss: 0.0098, Val Acc: 0.9984

======================================================================
🚀 Training CNN-BiLSTM on DATASET3 dataset
======================================================================
[dataset3] Epoch 1/3 | Batch 139/139 | Loss: 0.3497, Acc: 0.8435
[dataset3] Epoch 1/3 | Train Loss: 0.4518, Train Acc: 0.7863 | Val Loss: 0.3366, Val Acc: 0.8566
[dataset3] Epoch 2/3 | Batch 139/139 | Loss: 0.2770, Acc: 0.8722
[dataset3] Epoch 2/3 | Train Loss: 0.3166, Train Acc: 0.8633 | Val Loss: 0.2849, Val Acc: 0.8739
[dataset3] Epoch 3/3 | Batch 139/139 | Loss: 0.2620, Acc: 0.8855
[dataset3] Epoch 3/3 | Train Loss: 0.2841, Train Acc: 0.8752 | Val Loss: 0.2699, Val Acc: 0.8801

✅ All datasets trained successfully!

======================================================================
📈 Final Validation Summary (CNN-BiLSTM)
======================================================================
dataset1             | Val Acc: 0.9731 | Val Loss: 0.0860
dataset2             | Val Acc: 0.9984 | Val Loss: 0.0098
dataset3             | Val Acc: 0.8801 | Val Loss: 0.2699
```

In [ ]: