

ДОДАТОК А

Архітектури моделей глибокого навчання (PyTorch)

```

import torch
import torch.nn as nn

class CNN1D(nn.Module):
    """1-D згорткова мережа для класифікації мережевого
    трафіку"""
    def __init__(self, num_classes=15):
        super(CNN1D, self).__init__()
        self.features = nn.Sequential(
            nn.Conv1d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv1d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.AdaptiveMaxPool1d(1),
            nn.Flatten()
        )
        self.classifier = nn.Sequential(
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        return self.classifier(self.features(x))

class LSTMClassifier(nn.Module):
    """Класифікатор на базі рекурентної мережі LSTM"""
    def __init__(self, num_classes=15):
        super(LSTMClassifier, self).__init__()
        self.lstm = nn.LSTM(

```

```

        input_size=1,
        hidden_size=64,
        num_layers=2,
        batch_first=True,
        dropout=0.3
    )
    self.fc = nn.Linear(64, num_classes)

def forward(self, x):
    # x має розмірність (B, T, 1)
    out, _ = self.lstm(x)
    out = out[:, -1, :]
    return self.fc(out)

class CNNLSTMClassifier(nn.Module):
    """Гібридна згортково-рекурентна нейронна мережа (CNN-
    LSTM)"""
    def __init__(self, num_classes=15):
        super(CNNLSTMClassifier, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv1d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv1d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
        )
        self.lstm = nn.LSTM(
            input_size=64,
            hidden_size=64,
            num_layers=2,
            batch_first=True,
            dropout=0.3
        )
        self.fc = nn.Sequential(
            nn.Linear(64, 64),
            nn.ReLU(),

```

```

        nn.Dropout(0.3),
        nn.Linear(64, num_classes)
    )

    def forward(self, x):
        # Вхід x: (B, 1, F), де F - довжина вектора ознак
        x = self.conv(x)  # Вихід: (B, 64, F)
        x = x.transpose(1, 2)  # Транспонуємо до (B, F, 64)
для LSTM
        out, _ = self.lstm(x)  # Вихід: (B, F, 64)
        out = out[:, -1, :]  # Беремо останній прихований
стан: (B, 64)
        return self.fc(out)

```

ДОДАТОК Б

Процедури навчання моделей

```

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

def train_pytorch_model(model, train_loader, val_loader,
epochs=30, lr=0.001, patience=5, device="cpu"):
    """Процедура навчання нейронних мереж PyTorch з ранньою
зупинкою"""
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimiser = optim.Adam(model.parameters(), lr=lr)
    best_val_loss = float("inf")
    best_state = None
    no_improve = 0

    for epoch in range(1, epochs + 1):
        model.train()
        running_loss = 0.0
        for X_batch, y_batch in train_loader:
            X_batch, y_batch = X_batch.to(device),
y_batch.to(device)

            optimiser.zero_grad()
            logits = model(X_batch)
            loss = criterion(logits, y_batch)
            loss.backward()
            optimiser.step()
            running_loss += loss.item() * X_batch.size(0)

```

```

        # Валідація
        model.eval()
        val_loss = 0.0
        with torch.no_grad():
            for X_batch, y_batch in val_loader:
                X_batch, y_batch = X_batch.to(device),
y_batch.to(device)

                logits = model(X_batch)
                loss = criterion(logits, y_batch)
                val_loss += loss.item() * X_batch.size(0)

        val_loss /= len(val_loader.dataset)

        # Перевірка для ранньої зупинки
        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_state = {k: v.cpu().clone() for k, v in
model.state_dict().items()}
            no_improve = 0
        else:
            no_improve += 1
            if no_improve >= patience:
                print(f"Рання зупинка на епосі {epoch}")
                break

        if best_state is not None:
            model.load_state_dict(best_state)
        return model

def train_random_forest(X_train, y_train, X_test, y_test):
    """Навчання та оцінка моделі випадкового лісу"""
    clf = RandomForestClassifier(n_estimators=100,
random_state=42, n_jobs=-1)
    clf.fit(X_train, y_train)

```

```

        y_pred = clf.predict(X_test)
        print(f"Random          Forest          Точність:
{accuracy_score(y_test, y_pred):.4f}")
        return y_pred, clf

def train_svm(X_train, y_train, X_test, y_test,
max_samples=50000):
    """Навчання та оцінка методу опорних векторів з
    підсемплюванням"""
    if len(X_train) > max_samples:
        rng = np.random.RandomState(42)
        idx = rng.choice(len(X_train), size=max_samples,
replace=False)
        X_train, y_train = X_train[idx], y_train[idx]

    clf = SVC(kernel="rbf", C=1.0, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"SVM          Точність:          {accuracy_score(y_test,
y_pred):.4f}")
    return y_pred, clf

```

ДОДАТОК В

Програмний код візуалізації результатів

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_true, y_pred, label_names,
title, save_path):
    """Побудова та збереження теплової карти матриці
    неточностей"""
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(10, 8), dpi=150)
    sns.heatmap(
        cm, annot=True, fmt="d", cmap="YlOrRd",
        xticklabels=label_names,
        yticklabels=label_names, ax=ax,
        linewidths=0.5, linecolor="grey",
    )
    ax.set_title(title, fontsize=16)
    ax.set_xlabel("Передбачений клас", fontsize=13)
    ax.set_ylabel("Реальний клас", fontsize=13)
    plt.tight_layout()
    fig.savefig(save_path)
    plt.close(fig)
```

ДОДАТОК Г

Відомість кваліфікаційної роботи

[illegible]