ДОДАТОК А

Лістинг детектора аномалій на основі автоенкодера AnomalyDetector

```
class AnomalyDetector:
    def __init__(self, threshold=None, name="Autoencoder"):
        self.threshold = threshold
        self.scaler = StandardScaler()
        self.model = None
        self.name = name

    def fit(self, X):
        X_scaled = self.scaler.fit_transform(X)
        input_dim = X_scaled.shape[1]

        # Архітектура автоенкодера
        input_layer = Input(shape=(input_dim,))
        encoded = Dense(64, activation='relu')(input_layer)
        encoded = Dense(32, activation='relu')(encoded)
        decoded = Dense(64, activation='relu')(encoded)
        output_layer = Dense(input_dim,
activation='linear')(decoded)

        self.model = Model(input_layer, output_layer)
        self.model.compile(optimizer='adam', loss='mse')

        # Навчання з раннім зупиненням
        early_stop = EarlyStopping(monitor='loss', patience=5,
restore_best_weights=True)
        history = self.model.fit(
            X_scaled, X_scaled,
            epochs=50, batch_size=32,
            verbose=0, callbacks=[early_stop]
        )
```

```
        # Визначення порогу аномалій
        recon = self.model.predict(X_scaled, verbose=0)
        errors = np.mean(np.square(X_scaled - recon), axis=1)
        self.threshold = np.percentile(errors, 95) if
self.threshold is None else self.threshold

        return history

    def predict(self, X):
        X_scaled = self.scaler.transform(X)
        recon = self.model.predict(X_scaled, verbose=0)
        errors = np.mean(np.square(X_scaled - recon), axis=1)
        binary = (errors > self.threshold).astype(int)
        return binary, errors / (2 * self.threshold)
```

ДОДАТОК Б

Лістинг детектора на основі генеративно-змагальної мережі GANDetector

```
class GANDetector:
    def __init__(self, threshold=None, latent_dim=20,
name="GAN"):
        self.latent_dim = latent_dim
        self.threshold = threshold
        self.scaler = StandardScaler()
        self.name = name
        self.generator = None
        self.discriminator = None
        self.gan = None

    def _build_gan(self, input_dim):
        # Генератор
        g_input = Input(shape=(self.latent_dim,))
        g = Dense(64, activation='relu')(g_input)
        g = Dense(128, activation='relu')(g)
        g_output = Dense(input_dim, activation='tanh')(g)
        generator = Model(g_input, g_output)

        # Дискримінатор
        d_input = Input(shape=(input_dim,))
        d = Dense(128, activation='relu')(d_input)
        d = Dense(64, activation='relu')(d)
        d_output = Dense(1, activation='sigmoid')(d)
        discriminator = Model(d_input, d_output)
        discriminator.compile(loss='binary_crossentropy',
optimizer='adam')

        # Повна GAN модель
        discriminator.trainable = False
        gan_input = Input(shape=(self.latent_dim,))
        gan_output = discriminator(generator(gan_input))
```

```
        gan = Model(gan_input, gan_output)
        gan.compile(loss='binary_crossentropy',
optimizer='adam')

        return generator, discriminator, gan

    def fit(self, X, y, epochs=10, batch_size=32):
        X_scaled = self.scaler.fit_transform(X)
        normal = X_scaled[y == 0]  # Нормальні зразки
        input_dim = X_scaled.shape[1]

        # Перевірка, чи є нормальні зразки
        if normal.shape[0] == 0:
            print("Увага: відсутні зразки нормального класу
для навчання GAN")
            # Використання всіх даних, якщо немає нормального
класу
            normal = X_scaled

        # Ініціалізація моделей
        self.generator, self.discriminator, self.gan =
self._build_gan(input_dim)

        # Навчання GAN
        for epoch in range(epochs):
            # Вибір випадкових нормальних зразків
            idx = np.random.randint(0, normal.shape[0],
batch_size)
            real = normal[idx]

            # Генерація фейкових зразків
            noise = np.random.normal(0, 1, (batch_size,
self.latent_dim))
            fake = self.generator.predict(noise, verbose=0)

            # Навчання дискримінатора
```

```
            d_loss_real =
self.discriminator.train_on_batch(real, np.ones((batch_size,
1)))
            d_loss_fake =
self.discriminator.train_on_batch(fake, np.zeros((batch_size,
1)))

            # Навчання генератора
            g_loss = self.gan.train_on_batch(noise,
np.ones((batch_size, 1)))

            if (epoch + 1) % 5 == 0:
                print(f"Епоха {epoch+1}/{epochs}, G втрати:
{g_loss:.4f}, D втрати: {(d_loss_real + d_loss_fake)/2:.4f}")

        # Визначення порогу нормальності
        scores = self.discriminator.predict(normal,
verbose=0).flatten()
        self.threshold = np.percentile(scores, 5) if
self.threshold is None else self.threshold

    def predict(self, X):
        X_scaled = self.scaler.transform(X)
        scores = self.discriminator.predict(X_scaled,
verbose=0).flatten()
        binary = (scores < self.threshold).astype(int)
        return binary, np.abs(scores - self.threshold) /
self.threshold
```

ДОДАТОК В

Лістинг детектора на основі LightGBMDetector

```
class LightGBMDetector:
    def __init__(self, name="LightGBM"):
        self.model = None
        self.scaler = StandardScaler()
        self.name = name

    def fit(self, X, y):
        X_scaled = self.scaler.fit_transform(X)
        self.model = lgb.LGBMClassifier(
            n_estimators=100,
            random_state=42,
            objective='binary',
            verbose=-1
        )
        self.model.fit(X_scaled, y)

        # Отримання та виведення важливості ознак
        if hasattr(self.model, 'feature_importances_'):
            importances = self.model.feature_importances_
            indices = np.argsort(importances)[::-1]

            print(f"\n Топ-5 важливих ознак для {self.name}:")
            for i in range(min(5, len(importances))):
                print(f"   {i+1}. Ознака {indices[i]}:
{importances[indices[i]]:.4f}")

    def predict(self, X):
        X_scaled = self.scaler.transform(X)
        probs = self.model.predict_proba(X_scaled)[:, 1]
        preds = (probs > 0.5).astype(int)
        return preds, probs
```

ДОДАТОК Г

Лістинг гібридної системи виявлення вторгнень HybridIDS

```
class HybridIDS:
    def __init__(self, ensemble_mode="mean"):
        self.anomaly = AnomalyDetector()
        self.gan = GANDetector()
        self.lgbm = LightGBMDetector()
        self.ensemble_mode = ensemble_mode
        self.detectors = [self.anomaly, self.gan, self.lgbm]
        self.name = "HybridIDS"

    def fit(self, X, y):
        print("\n🔧 Навчання детектора аномалій...")
        self.anomaly.fit(X)

        print("\n Навчання GAN детектора...")
        self.gan.fit(X, y)

        print("\n Навчання LightGBM детектора...")
        self.lgbm.fit(X, y)

        print("\n Всі детектори навчено успішно!")

    def get_scores(self, X):
        _, s1 = self.anomaly.predict(X)
        _, s2 = self.gan.predict(X)
        _, s3 = self.lgbm.predict(X)

        if self.ensemble_mode == "median":
            return np.median(np.vstack([s1, s2, s3]), axis=0)
        else:  # mean
            return np.mean(np.vstack([s1, s2, s3]), axis=0)

    def predict(self, X, threshold=0.5):
```

```
        scores = self.get_scores(X)
        return (scores > threshold).astype(int), scores
```

ДОДАТОК Д

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	73 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Щербак В.О.			Розробка системи виявлення аномалій у кібербезпеці на основі машинного навчання	Шифр групи	Код напр./спец.
Перевірив		Бродецький Ф.А.				ІТШІпз-23-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					