ПОРІВНЯЛЬНИЙ АНАЛІЗ СУЧАСНИХ АЛГОРИТМІВ ТА ФРЕЙМВОРКІВ ДЛЯ QUANTUM MACHINE LEARNING

ПІДГОТУВАЛА ДОПОВІДЬ: РЕУЦЬКА АРИНА ОЛЕГІВНА

НАУКОВИЙ КЕРІВНИК: СТ. ВИКЛАДАЧ ГОРОХОВСЬКИЙ КИРИЛО СЕМЕНОВИЧ

ВСТУП

- Актуальність: автоматична класифікація модуляції є важливою для сучасних систем зв’язку, а квантове машинне навчання розглядається як перспективний підхід до аналізу складних сигналів
- Проблематика: різні QML-фреймворки мають різні можливості, але їх практична ефективність у задачах обробки радіосигналів ще недостатньо досліджена
- Мета роботи: порівняти сучасні QML-фреймворки та перевірити їх придатність до задачі класифікації модуляції радіосигналів

ТЕОРЕТИЧНА ОСНОВА QML

Quantum Machine Learning поєднує квантові обчислення та методи машинного навчання для аналізу складних даних.

Основні підходи:

- кодування класичних ознак у квантові стани;
- використання квантових ядер;
- побудова варіаційних квантових схем;
- застосування гібридних квантово-класичних моделей.

QML У ЗАДАЧАХ ОБРОБКИ РАДІОСИГНАЛІВ

Радіосигнали мають складну структуру, оскільки містять амплітудні, фазові, частотні та часові характеристики. QML є актуальним для таких задач через:

- високу розмірність сигналових даних;
- наявність шуму та змінного SNR;
- потребу в компактних моделях;
- можливість формування складних ознакових просторів;
- застосування гібридних моделей на сучасних NISQ-пристроях.

ОСНОВНІ ОБМЕЖЕННЯ QML

Практичне використання QML обмежується такими чинниками:

- складність кодування класичних даних у квантові стани;
- шум і обмежена кількість кубітів у NISQ-пристроях;
- залежність результатів від архітектури анзаців;
- проблема стабільності навчання;
- різна продуктивність програмних фреймворків.

ПОРІВНЯЛЬНИЙ АНАЛІЗ ОСНОВНИХ ФРЕЙМВОРКІВ QML

КРИТЕРІЇ ДОСЛІДЖЕННЯ

Критерій	Зміст критерію	Операціоналізація в експерименті	Практичне значення
Алгоритмічна репрезентативність	Наскільки фреймворк підтримує kernel-based, варіаційні або гібридні квантумні моделі	Аналіз архітектури API, типу моделі та способу побудови квантової частини	Визначає, чи придатне середовище для дослідження різних класів QML-алгоритмів
Інтеграція з класичним ML	Рівень сумісності з NumPy, scikit-learn, TensorFlow, PyTorch і тд.	Оцінка того, чи фреймворк дозволяє природно будувати гібридний пайплайн	Важливо для роботи з ознаками, попередньою обробкою та класифікатором
Прозорість навчання	Можливість відстежувати функцію витрат, норми градієнта, зміну параметрів та поведінку моделі в часі	Аналіз наявності графіків витрат, норм градієнта, історії епох та структури ознак тренування	Для оцінення як фінальної метрики, так і динаміки оптимізації

КРИТЕРІЇ ДОСЛІДЖЕННЯ

Критерій	Зміст критерію	Операціоналізація в експерименті	Практичне значення
Якість класифікації	Узагальнена здатність моделі розділяти класи на тестовій вибірці	Використання accuracy, balanced accuracy, f1-score, AUC, матриці помилок	Безпосередня оцінка прикладної придатності реалізації
Обчислювальна вартість	Витрати часу на навчання та запуск серії експериментів	Порівняння середнього train_time_s	Визначення придатності до серійного бенчмарку та повторних запусків
Стабільність середовища	Наскільки легко відтворити запуск без конфліктів залежностей та технічних збоїв	Аналіз практичної поведінки середовища під час реального тестування	Впливає на відтворюваність, масштабованість та надійність результатів

ІНФОРМАЦІЯ ПРО ДАТАСЕТ

```
Файл датасету: /kaggle/input/radioml2016-deepsigcom/RML2016.10a_dict.pkl
Кількість ключів у словнику: 220
Кількість доступних модуляцій: 11
Список модуляцій: ['8PSK', 'AM-DSB', 'AM-SSB', 'BPSK', 'CPFSK', 'GFSK', 'PAM4', 'QAM16', 'QAM64', 'QPSK', 'WBFM']
Список SNR: [-20, -18, -16, -14, -12, -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
Форма X після підготовки: (720, 8)
Форма y: (720,)
Баланс класів: [360 360]
Унікальні SNR у вибірці: [0, 10, 18]
======================================================================
Завантажено прикладів: 720
Форма X після PCA: (720, 8)
Баланс класів: [360 360]
Сумарна пояснена дисперсія PCA: 0.8872
```

RadioML
2016.10A

ФРЕЙМВОРКИ, ЩО ТЕСТУВАЛИСЯ

- Qibo
- PennyLane
- Qulacs
- TensorFlow Quantum
- TorchQuantum
- Amazon Braket

ІНСТРУМЕНТИ ДЛЯ ПРОВДЕННЯ ДОСЛІДЖЕННЯ

- Середовище Google Colab
- kagglehub
- matplotlib
- seaborn
- scikit-learn
- PyTorch

ПОРІВНЯЛЬНА ТАБЛИЦЯ

Фреймворк	Архітектурний тип	Інтеграція з класичним ML	Прозорість навчання	Середній bacc	Середній AUC	Середній час, с	Стабільність середовища
Qibo	Гібридний квантовий екстрактор ознак + класичний класифікатор	Висока	Середня	0.819	0.905	1.192	Висока
TorchQuantum	End-to-end тренувальна гібридна модель у стилі PyTorch	Висока	Висока	0.808	0.904	1.993	Середня

ПОРІВНЯЛЬНА ТАБЛИЦЯ

Фреймворк	Архітектурний тип	Інтеграція з класичним ML	Прозорість навчання	Середній bacc	Середній AUC	Середній час, с	Стабільність середовища
Amazon Braket	Квантовий екстрактор ознак у локальному симуляторі з інфраструктурною орієнтацією	Середня	Середня	0.781	0.873	6.782	Середня після ізоляції середовища
Qulacs	Високопродуктивний симулятор параметризованих систем + класичний класифікатор	Середня	Середня	0.767	0.865	1.134	Висока

ПОРІВНЯЛЬНА ТАБЛИЦЯ

Фреймворк	Архітектурний тип	Інтеграція з класичним ML	Прозорість навчання	Середній bacc	Середній AUC	Середній час, с	Стабільність середовища
TensorFlow Quantum	Гібридна квантумо-класична модель у графі TensorFlow	Висока	Висока	0.753	0.798	3.361	Низька
PennyLane	Диференційоване квантове програмування та варіаційна end-to-end модель	Висока	Найвища	0.672	0.787	64.095	Висока

РЕАЛІЗАЦІЯ ЗАДАЧІ ОБРОБКИ РАДІОСИГНАЛІВ

ПІДГОТОВКА ВХІДНИХ ДАНИХ

```
Тип: <class 'dict'>
Кількість ключів: 220
Модуляції: ['8PSK', 'AM-DSB', 'AM-SSB', 'BPSK', 'CPFSK', 'GFSK', 'PAM4', 'QAM16', 'QAM64', 'QPSK', 'WBFM']
SNR: [-20, -18, -16, -14, -12, -10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
Приклад ключа: ('QPSK', 2)
Форма масиву: (1000, 2, 128)
```

```
Форма X: (1440, 2, 128)
Форма y: (1440,)
Форма snr: (1440,)

Розподіл по класах:
BPSK 360
QPSK 360
8PSK 360
QAM16 360

Розподіл по SNR:
0 480
10 480
18 480
```

```
Форма матриці ознак: (1440, 50)
```

ПІДГОТОВКА ВХІДНИХ ДАНИХ

```python
def make_joint_strat_labels(y: np.ndarray, snr: np.ndarray) -> np.ndarray:
    return np.array([f"{yy}_{ss}" for yy, ss in zip(y, snr)])


def split_data(F, y, snr, cfg):
    strat = make_joint_strat_labels(y, snr)

    F_trainval, F_test, y_trainval, y_test, snr_trainval, snr_test = train_test_split(
        F, y, snr,
        test_size=cfg.test_size,
        random_state=cfg.seed,
        stratify=strat
    )

    strat_trainval = make_joint_strat_labels(y_trainval, snr_trainval)
    relative_val = cfg.val_size / (1.0 - cfg.test_size)

    F_train, F_val, y_train, y_val, snr_train, snr_val = train_test_split(
        F_trainval, y_trainval, snr_trainval,
        test_size=relative_val,
        random_state=cfg.seed,
        stratify=strat_trainval
    )

    print("Train:", F_train.shape)
    print("Val:", F_val.shape)
    print("Test:", F_test.shape)

    return F_train, F_val, F_test, y_train, y_val, y_test, snr_train, snr_val, snr_test


F_train_raw, F_val_raw, F_test_raw, y_train, y_val, y_test, snr_train, snr_val, snr_test = split_data(
    F_sub, y_sub, snr_sub, CFG
)
```

```
Train: (1008, 50)
Val: (216, 50)
Test: (216, 50)
```

ПІДГОТОВКА ВХІДНИХ ДАНИХ

```
feature_scaler = StandardScaler()
F_train_scaled = feature_scaler.fit_transform(F_train_raw).astype(np.float32)
F_val_scaled = feature_scaler.transform(F_val_raw).astype(np.float32)
F_test_scaled = feature_scaler.transform(F_test_raw).astype(np.float32)

pca = PCA(n_components=CFG.n_pca_components, random_state=CFG.seed)
X_train = pca.fit_transform(F_train_scaled).astype(np.float32)
X_val = pca.transform(F_val_scaled).astype(np.float32)
X_test = pca.transform(F_test_scaled).astype(np.float32)

snr_train_feat = (snr_train.astype(np.float32) / 20.0).reshape(-1, 1)
snr_val_feat = (snr_val.astype(np.float32) / 20.0).reshape(-1, 1)
snr_test_feat = (snr_test.astype(np.float32) / 20.0).reshape(-1, 1)

print("X_train:", X_train.shape)
print("X_val:", X_val.shape)
print("X_test:", X_test.shape)
print("F_train_scaled:", F_train_scaled.shape)
print("Пояснена дисперсія PCA:", float(np.sum(pca.explained_variance_ratio_)))
```

```
X_train: (1008, 8)
X_val: (216, 8)
X_test: (216, 8)
F_train_scaled: (1008, 50)
Пояснена дисперсія PCA: 0.8096533417701721
```

ПІДГОТОВКА ВХІДНИХ ДАНИХ

```python
class AMCDataset(Dataset):
    def __init__(self, X_pca: np.ndarray, X_raw_scaled: np.ndarray, y: np.ndarray, snr: np.ndarray):
        self.X_pca = torch.tensor(X_pca, dtype=torch.float32)
        self.X_raw = torch.tensor(X_raw_scaled, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)
        self.snr = torch.tensor((snr.astype(np.float32) / 20.0).reshape(-1, 1), dtype=torch.float32)

    def __len__(self):
        return len(self.X_pca)

    def __getitem__(self, idx):
        return self.X_pca[idx], self.X_raw[idx], self.snr[idx], self.y[idx]


train_ds = AMCDataset(X_train, F_train_scaled, y_train, snr_train)
val_ds = AMCDataset(X_val, F_val_scaled, y_val, snr_val)
test_ds = AMCDataset(X_test, F_test_scaled, y_test, snr_test)

train_loader = DataLoader(train_ds, batch_size=CFG.batch_size, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=CFG.batch_size, shuffle=False)
test_loader = DataLoader(test_ds, batch_size=CFG.batch_size, shuffle=False)
```

TORCHQUANTUM

```python
class TorchQuantumFineTuned(nn.Module):
    def __init__(
        self,
        n_qubits=8,
        n_layers=3,
        n_reuploads=2,
        n_classes=4,
        n_raw_features=32,
        hidden_dim1=64,
        hidden_dim2=32,
        dropout=0.15
    ):
        super().__init__()
        self.n_qubits = n_qubits
        self.n_layers = n_layers
        self.n_reuploads = n_reuploads

        self.input_norm = nn.LayerNorm(n_qubits)

        self.alpha = nn.Parameter(torch.ones(n_reuploads, n_qubits))
        self.theta_rx = nn.Parameter(0.1 * torch.randn(n_reuploads, n_layers, n_qubits))
        self.theta_ry = nn.Parameter(0.1 * torch.randn(n_reuploads, n_layers, n_qubits))
        self.theta_rz = nn.Parameter(0.1 * torch.randn(n_reuploads, n_layers, n_qubits))

        self.measure = tq.MeasureAll(tq.PauliZ)

        head_in_dim = n_qubits + n_qubits + n_raw_features + 1

        self.head = nn.Sequential(
            nn.Linear(head_in_dim, hidden_dim1),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim1, hidden_dim2),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim2, n_classes)
        )
```

```python
def forward(self, x_pca, x_raw, snr_feat):
    bsz = x_pca.shape[0]
    x_enc = torch.tanh(self.input_norm(x_pca))

    qdev = tq.QuantumDevice(n_wires=self.n_qubits, bsz=bsz, device=str(x_pca.device))

    for r in range(self.n_reuploads):
        for i in range(self.n_qubits):
            tqf.ry(qdev, wires=i, params=(0.75 * self.alpha[r, i] * x_enc[:, i]))

        for l in range(self.n_layers):
            for i in range(self.n_qubits - 1):
                tqf.cnot(qdev, wires=[i, i + 1])
            tqf.cnot(qdev, wires=[self.n_qubits - 1, 0])

            for i in range(self.n_qubits):
                tqf.rx(qdev, wires=i, params=self.theta_rx[r, l, i].expand(bsz))
                tqf.ry(qdev, wires=i, params=self.theta_ry[r, l, i].expand(bsz))
                tqf.rz(qdev, wires=i, params=self.theta_rz[r, l, i].expand(bsz))

    z = self.measure(qdev)
    if len(z.shape) > 2:
        z = z.view(z.shape[0], -1)

    fused = torch.cat([z, x_pca, x_raw, snr_feat], dim=1)
    logits = self.head(fused)
    return logits
```

QIBO

```python
class QiboFeatureExtractor:
    def __init__(self, n_qubits=8, n_layers=3, seed=42):
        self.n_qubits = n_qubits
        self.n_layers = n_layers

        rng = np.random.default_rng(seed)
        self.theta_rx = 0.1 * rng.normal(size=(n_layers, n_qubits))
        self.theta_ry = 0.1 * rng.normal(size=(n_layers, n_qubits))
        self.theta_rz = 0.1 * rng.normal(size=(n_layers, n_qubits))

    def _forward_one(self, x: np.ndarray) -> np.ndarray:
        c = Circuit(self.n_qubits)

        for i in range(self.n_qubits):
            c.add(gates.RY(i, float(np.tanh(x[i]))))

        for l in range(self.n_layers):
            for i in range(self.n_qubits - 1):
                c.add(gates.CNOT(i, i + 1))
            c.add(gates.CNOT(self.n_qubits - 1, 0))

            for i in range(self.n_qubits):
                c.add(gates.RX(i, float(self.theta_rx[l, i])))
                c.add(gates.RY(i, float(self.theta_ry[l, i])))
                c.add(gates.RZ(i, float(self.theta_rz[l, i])))

        result = c()
        state = result.state()
        z = expect_z_all_from_state(state, self.n_qubits)
        return z

    def transform(self, X: np.ndarray, verbose: bool = True) -> Tuple[np.ndarray, float]:
        feats = []
        t0 = time.time()

        for k, x in enumerate(X):
            feats.append(self._forward_one(x))
            if verbose and (k + 1) % 200 == 0:
                print(f"Оброблено {k+1}/{len(X)} прикладів")

        feats = np.array(feats, dtype=np.float32)
        dt = time.time() - t0
        print(f"Qibo quantum feature extraction done in {dt:.2f} s")
        return feats, dt
```

```python
class QiboFineTuned:
    def __init__(self, cfg):
        self.cfg = cfg
        self.extractor = QiboFeatureExtractor(
            n_qubits=cfg.n_qubits,
            n_layers=cfg.n_layers,
            seed=cfg.seed
        )
        self.q_scaler = StandardScaler()
        self.head = None
        self.history = None
        self.train_time_s = None
        self.feature_time_trainval_s = None

    def _build_hybrid(self, X_pca, F_scaled, X_q, snr_feat, fit_q_scaler=False):
        if fit_q_scaler:
            X_qs = self.q_scaler.fit_transform(X_q)
        else:
            X_qs = self.q_scaler.transform(X_q)

        X_hybrid = np.concatenate(
            [F_scaled, X_pca, X_qs, snr_feat.astype(np.float32)],
            axis=1
        ).astype(np.float32)

        return X_hybrid

    def fit(
        self,
        X_train_pca, F_train_scaled, snr_train_feat, y_train,
        X_val_pca, F_val_scaled, snr_val_feat, y_val,
        verbose=True
    ):
        X_train_q, t_train = self.extractor.transform(X_train_pca, verbose=verbose)
        X_val_q, t_val = self.extractor.transform(X_val_pca, verbose=False)

        self.feature_time_trainval_s = t_train + t_val

        X_train_h = self._build_hybrid(X_train_pca, F_train_scaled, X_train_q, snr_train_feat, fit_q_scaler=True)
        X_val_h = self._build_hybrid(X_val_pca, F_val_scaled, X_val_q, snr_val_feat, fit_q_scaler=False)

        clf = MLPClassifier(
            hidden_layer_sizes=self.cfg.hidden_dims,
            activation="relu",
            solver="adam",
            alpha=self.cfg.weight_decay,
            batch_size=self.cfg.batch_size,
            learning_rate_init=self.cfg.learning_rate,
            max_iter=1,
            warm_start=True,
            random_state=self.cfg.seed
        )

        classes = np.arange(len(self.cfg.target_classes))
```

QIBO

```python
best_model = None
best_val_f1 = -np.inf
patience_counter = 0

history = {
    "train_loss": [],
    "val_loss": [],
    "train_f1": [],
    "val_f1": [],
}

t0 = time.time()

for epoch in range(self.cfg.epochs):
    if epoch == 0:
        clf.partial_fit(X_train_h, y_train, classes=classes)
    else:
        clf.partial_fit(X_train_h, y_train)

    y_train_pred = clf.predict(X_train_h)
    y_train_proba = clf.predict_proba(X_train_h)
    train_f1 = f1_score(y_train, y_train_pred, average="macro")
    train_loss = log_loss(y_train, y_train_proba, labels=classes)

    y_val_pred = clf.predict(X_val_h)
    y_val_proba = clf.predict_proba(X_val_h)
    val_f1 = f1_score(y_val, y_val_pred, average="macro")
    val_loss = log_loss(y_val, y_val_proba, labels=classes)

    history["train_loss"].append(float(train_loss))
    history["val_loss"].append(float(val_loss))
    history["train_f1"].append(float(train_f1))
    history["val_f1"].append(float(val_f1))
```

```python
        print(
            f"[QiboFineTuned] Epoch {epoch+1:02d}/{self.cfg.epochs} | "
            f"train_loss={train_loss:.4f} | val_loss={val_loss:.4f} | "
            f"train_f1={train_f1:.4f} | val_f1={val_f1:.4f}"
        )

        if val_f1 > best_val_f1:
            best_val_f1 = val_f1
            best_model = copy.deepcopy(clf)
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= self.cfg.early_stopping_patience:
                print("[QiboFineTuned] Early stopping")
                break

    self.train_time_s = time.time() - t0
    self.head = best_model
    self.history = history
```

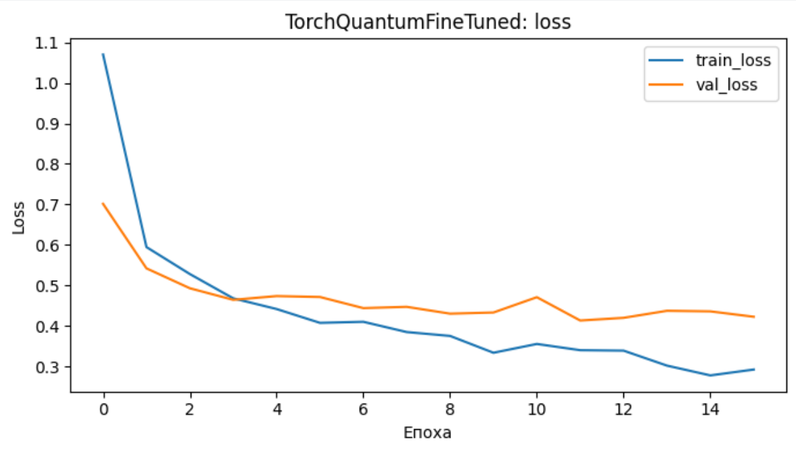
РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ TORCHQUANTUMFINETUNED

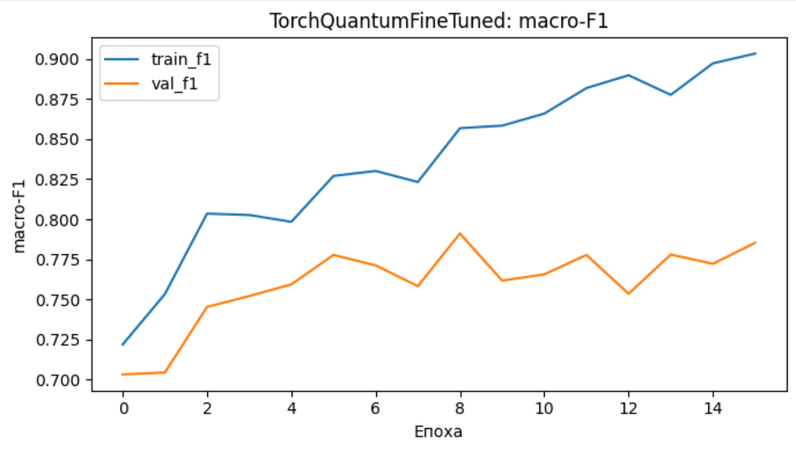
```
[TorchQuantumFineTuned] Загальні метрики:
{
  "acc": 0.7777777777777778,
  "bacc": 0.7777777777777777,
  "f1_macro": 0.776694256483905,
  "auc_macro_ovr": 0.9415580704160951
}

Classification report:
              precision    recall  f1-score   support

        BPSK     0.9434    0.9259    0.9346        54
        QPSK     0.5694    0.7593    0.6508        54
        8PSK     0.6667    0.4815    0.5591        54
       QAM16     0.9808    0.9444    0.9623        54

    accuracy                         0.7778       216
   macro avg     0.7901    0.7778    0.7767       216
weighted avg     0.7901    0.7778    0.7767       216
```

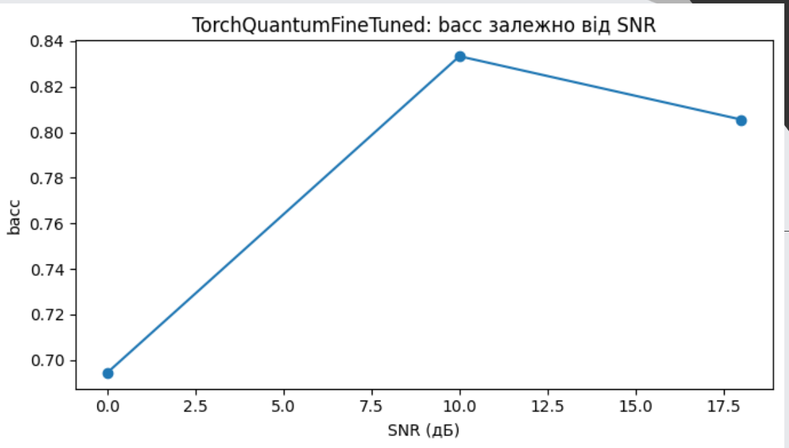
	snr_db	acc	bacc	f1_macro	auc_macro_ovr
0	0	0.694444	0.694444	0.704058	0.906379
1	10	0.833333	0.833333	0.831461	0.950360
2	18	0.805556	0.805556	0.792984	0.961163

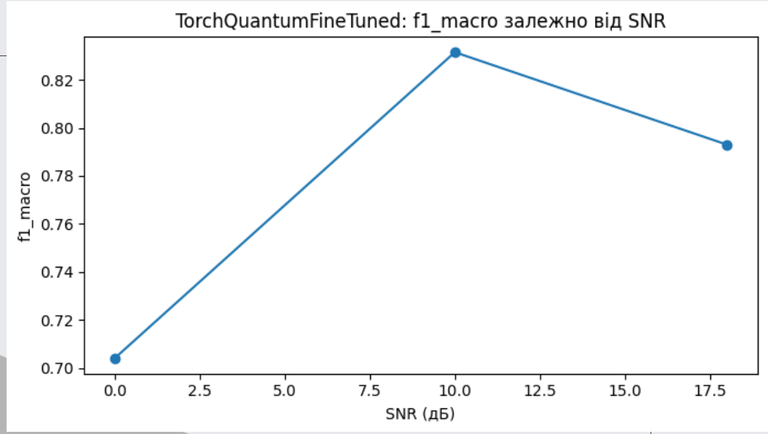
```
Загальний час навчання: 162.8277142047882
```

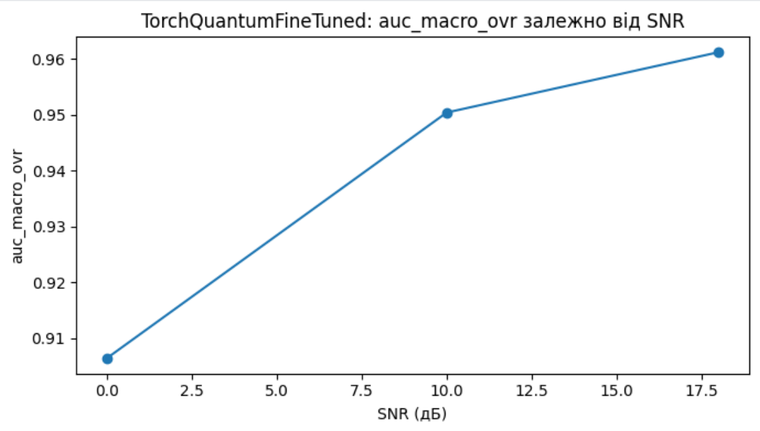
РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ TORCHQUANTUMFINETUNED



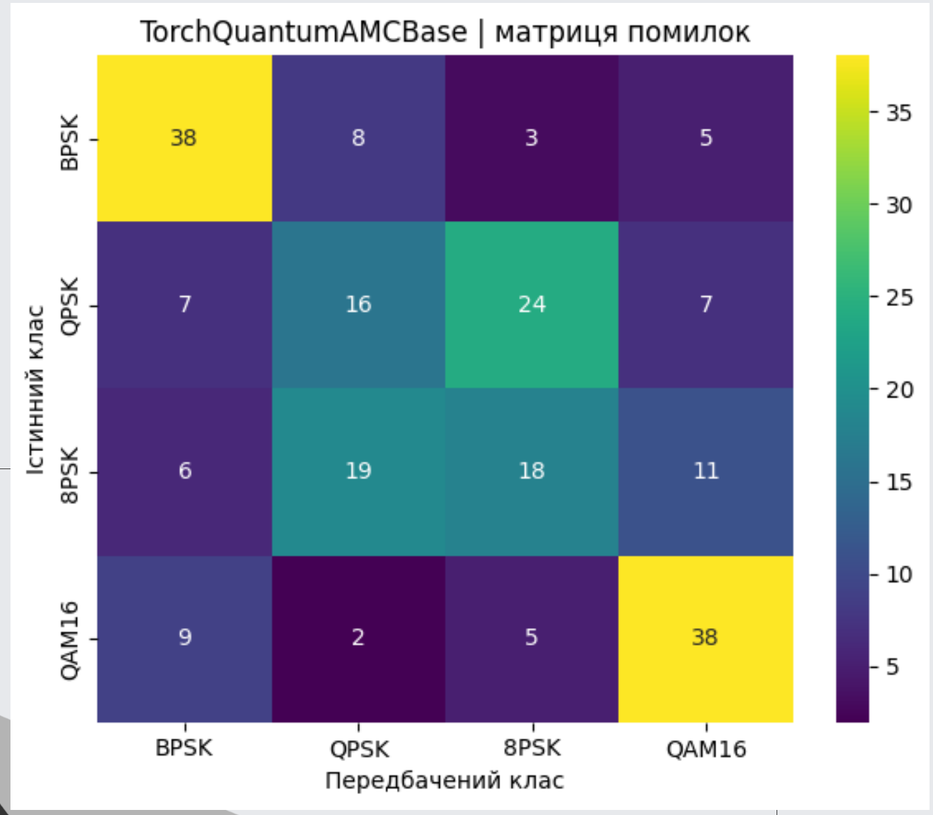








РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ TORCHQUANTUMFINETUNED



Найбільша кількість хибних класифікацій припадала на змішування QPSK та 8PSK, причому клас 8PSK досить часто помилково відносився до QPSK.

РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ QIBOQUANTUMFINETUNED

```
[QiboFineTuned] Загальні метрики:
{
  "acc": 0.7916666666666666,
  "bacc": 0.7916666666666667,
  "f1_macro": 0.7942464091748683,
  "auc_macro_ovr": 0.9441015089163237
}
```

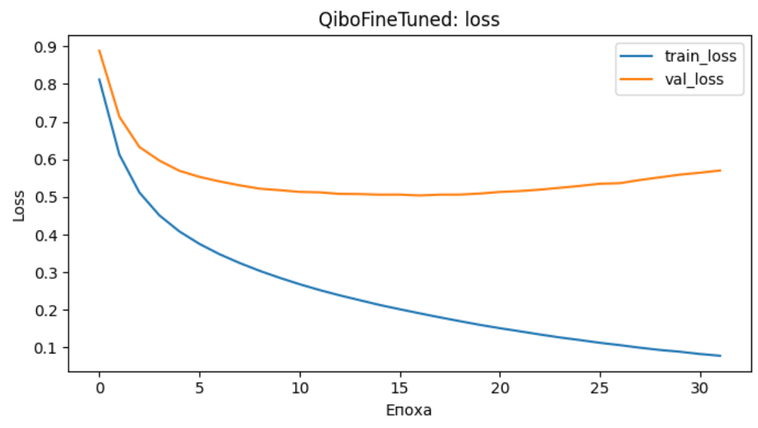
Classification report:

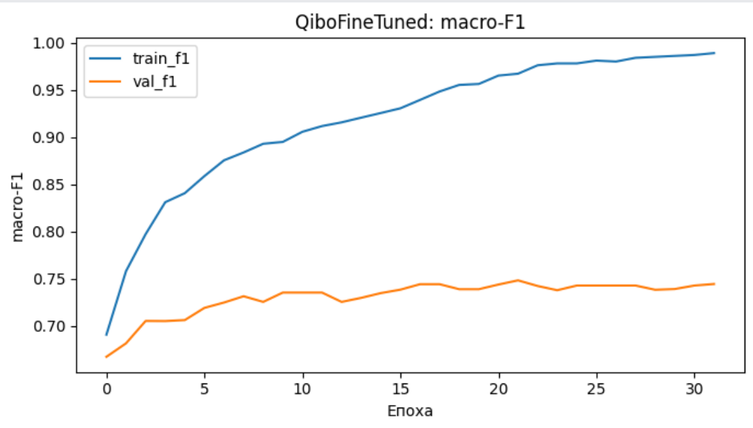
	precision	recall	f1-score	support
BPSK	0.9608	0.9074	0.9333	54
QPSK	0.6415	0.6296	0.6355	54
8PSK	0.6271	0.6852	0.6549	54
QAM16	0.9623	0.9444	0.9533	54
accuracy			0.7917	216
macro avg	0.7979	0.7917	0.7942	216
weighted avg	0.7979	0.7917	0.7942	216

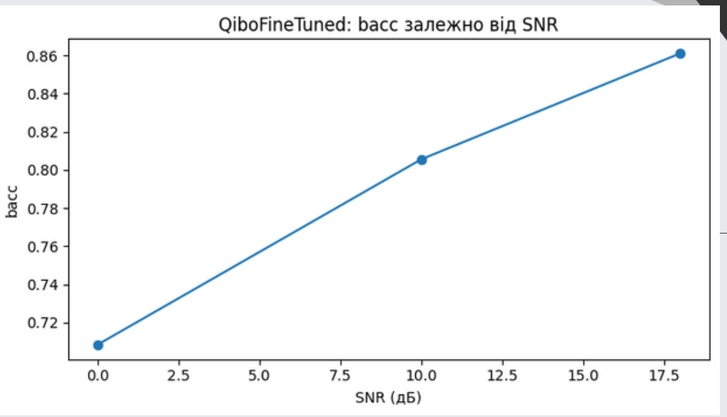
	snr_db	acc	bacc	f1_macro	auc_macro_ovr
0	0	0.708333	0.708333	0.712355	0.892747
1	10	0.805556	0.805556	0.809524	0.947788
2	18	0.861111	0.861111	0.859888	0.972479

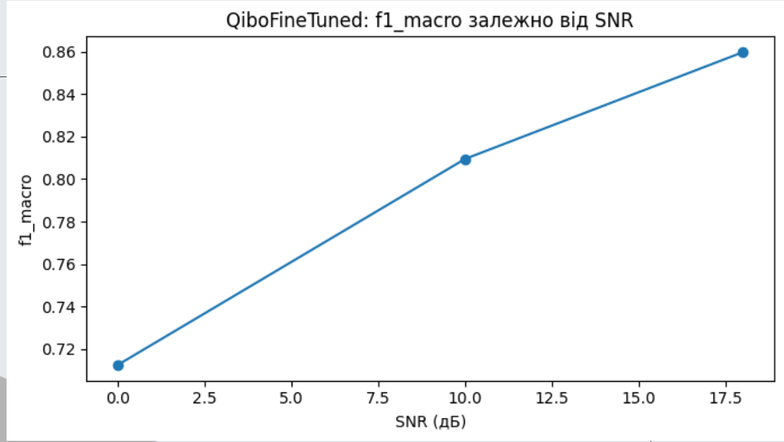
Загальний час навчання: 1.465602159500122
Час побудови квантових ознак (train+val): 5.53201699256897

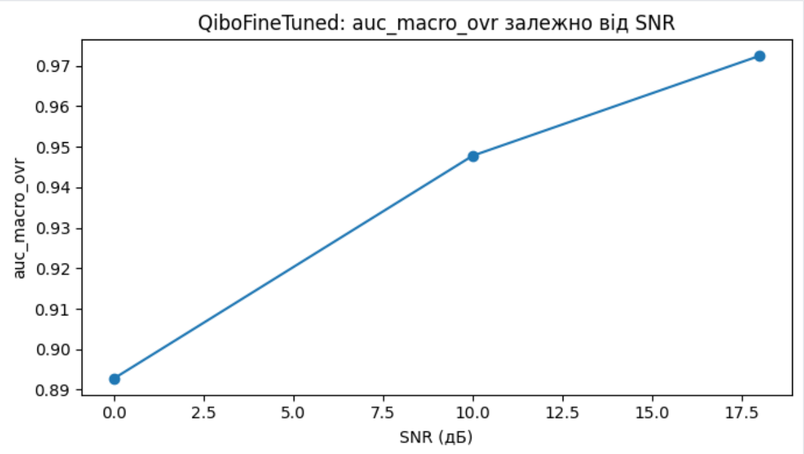
РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ QIBOQUANTUMFINETUNED



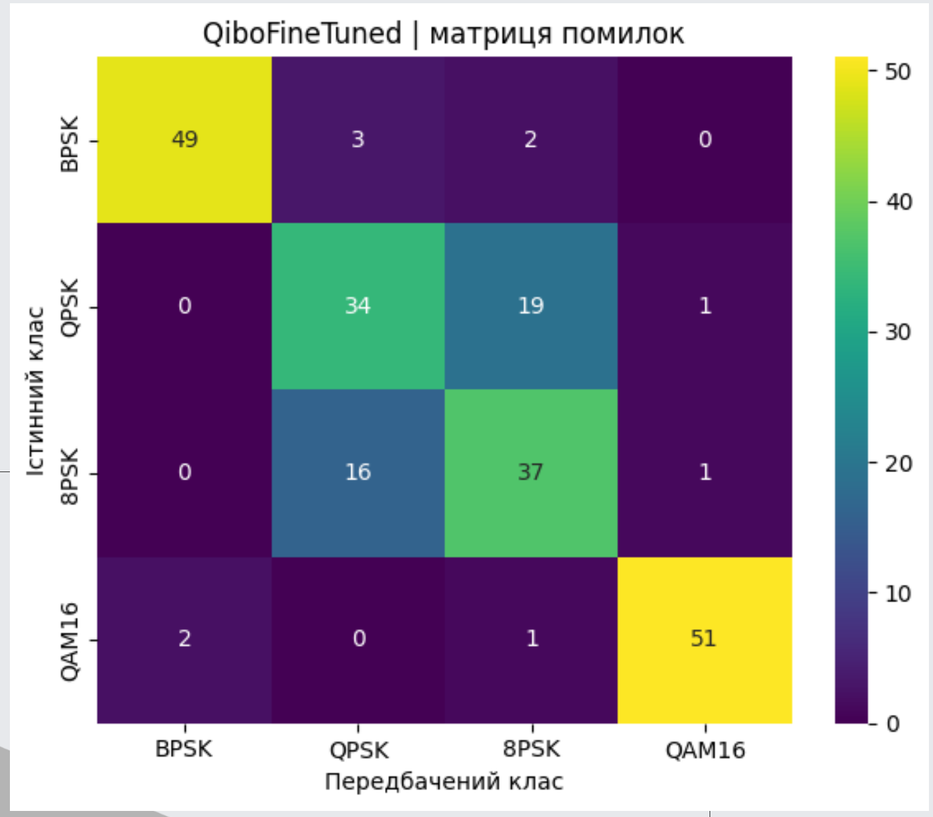








РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ QIBOQUANTUMFINETUNED



Схожа тенденція з TorchQuantumFineTuned, однак розподіл помилок був більш рівномірним, а кількість правильних класифікацій для 8PSK є вищою.

РЕЗУЛЬТАТИ НАВЧАННЯ МОДЕЛІ TORCHQUANTUMFINETUNED

```
[TorchQuantumFineTuned] Загальні метрики:
{
  "acc": 0.7777777777777778,
  "bacc": 0.7777777777777777,
  "f1_macro": 0.776694256483905,
  "auc_macro_ovr": 0.9415580704160951
}

Classification report:
              precision    recall  f1-score   support

        BPSK     0.9434    0.9259    0.9346        54
        QPSK     0.5694    0.7593    0.6508        54
        8PSK     0.6667    0.4815    0.5591        54
       QAM16     0.9808    0.9444    0.9623        54

    accuracy                         0.7778       216
   macro avg     0.7901    0.7778    0.7767       216
weighted avg     0.7901    0.7778    0.7767       216
```

	snr_db	acc	bacc	f1_macro	auc_macro_ovr
0	0	0.694444	0.694444	0.704058	0.906379
1	10	0.833333	0.833333	0.831461	0.950360
2	18	0.805556	0.805556	0.792984	0.961163

```
Загальний час навчання: 162.8277142047882
```

ДАТАСЕТ ДЛЯ ЗОВНІШНЬОГО ТЕСТУВАННЯ

```
Завантажую RadioML 2016.10B через kagglehub ...
Using Colab cache for faster access to the 'rml201610b' dataset.
Датасет завантажено у: /kaggle/input/rml201610b
Знайдені кандидати:
  /kaggle/input/rml201610b/RML2016.10b.dat
Використовується файл: /kaggle/input/rml201610b/RML2016.10b.dat
Тип об'єкта: <class 'dict'>
Кількість ключів: 200
Форма зовнішніх IQ-даних: (600, 2, 128)
Форма y_ext: (600,)
Форма snr_ext: (600,)

Розподіл по класах:
BPSK 150
QPSK 150
8PSK 150
QAM16 150

Розподіл по SNR:
0 200
10 200
18 200
Форма матриці ознак: (600, 50)

ПІДГОТОВКА ЗОВНІШНЬОГО НАБОРУ ЗАВЕРШЕНА
F_ext_scaled: (600, 50)
X_ext_pca: (600, 8)
snr_ext_feat: (600, 1)
```

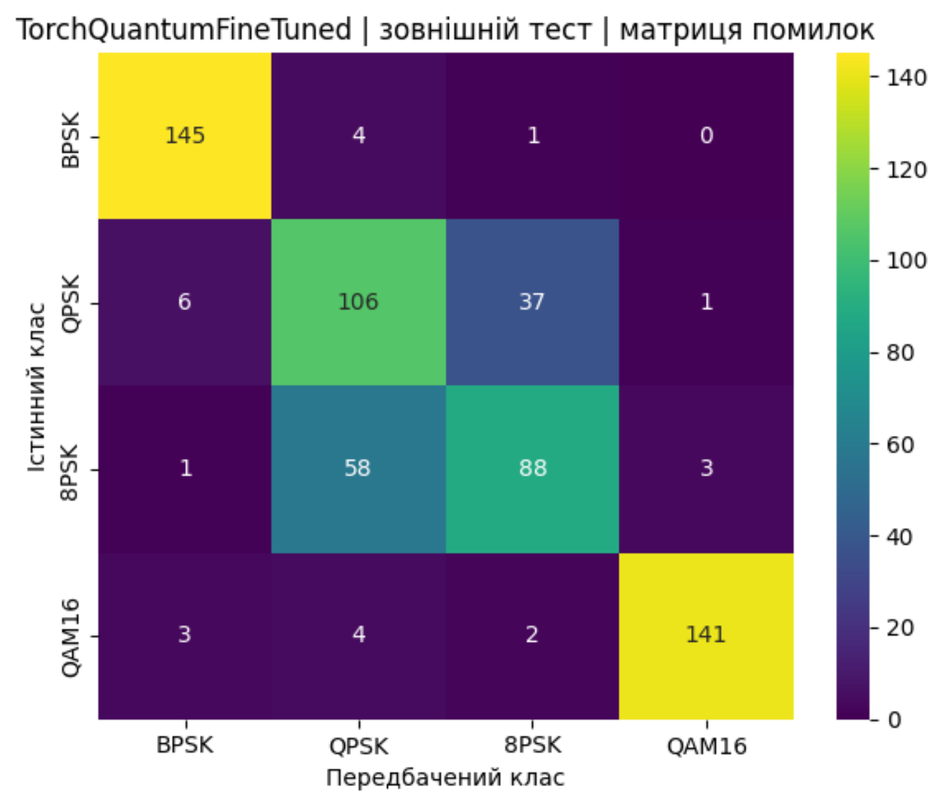
Датасет RadioML 2016.10B

РЕЗУЛЬТАТИ ЗОВНІШНЬОГО ТЕСТУВАННЯ (TORCHQUANTUMFINETUNED)

```
[TorchQuantumFineTuned | External 10B] Загальні метрики:
{
  "acc": 0.8,
  "bacc": 0.7999999999999999,
  "f1_macro": 0.799557623467134,
  "auc_macro_ovr": 0.9456703703703704
}

Classification report:
              precision    recall  f1-score   support

        BPSK     0.9355    0.9667    0.9508       150
        QPSK     0.6163    0.7067    0.6584       150
        8PSK     0.6875    0.5867    0.6331       150
       QAM16     0.9724    0.9400    0.9559       150

    accuracy                         0.8000       600
   macro avg     0.8029    0.8000    0.7996       600
weighted avg     0.8029    0.8000    0.7996       600
```

	snr_db	acc	bacc	f1_macro	auc_macro_ovr
0	0	0.755	0.755	0.756957	0.911733
1	10	0.810	0.810	0.806092	0.955900
2	18	0.835	0.835	0.834431	0.961600

РЕЗУЛЬТАТИ ЗОВНІШНЬОГО ТЕСТУВАННЯ (TORCHQUANTUMFINETUNED)



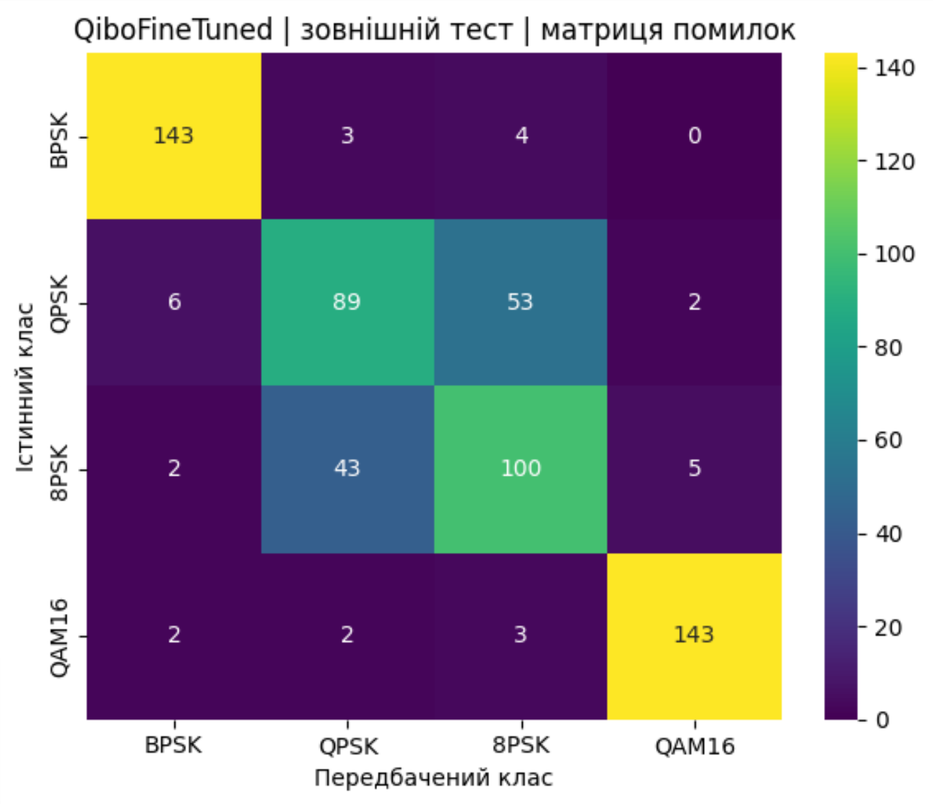
Найкраще розпізнавалися класи BPSK та QAM16, для яких кількість правильних класифікацій становила 145 та 141 відповідно з 150 можливих. Найбільша частина похибок зосереджена між класами QPSK та 8PSK, оскільки QPSK у 37 випадках було віднесено до 8PSK, а 8PSK у 58 випадках – до QPSK .

РЕЗУЛЬТАТИ ЗОВНІШНЬОГО ТЕСТУВАННЯ (QIBOQUANTUMFINETUNED)

```
Qibo quantum feature extraction done in 2.70 s
Qibo quantum feature extraction done in 2.74 s

[QiboFineTuned | External 10B] Загальні метрики:
{
  "acc": 0.7916666666666666,
  "bacc": 0.7916666666666667,
  "f1_macro": 0.7906495180820767,
  "auc_macro_ovr": 0.9473259259259259
}

Classification report:
              precision    recall  f1-score   support

        BPSK     0.9346    0.9533    0.9439       150
        QPSK     0.6496    0.5933    0.6202       150
        8PSK     0.6250    0.6667    0.6452       150
       QAM16     0.9533    0.9533    0.9533       150

    accuracy                         0.7917       600
   macro avg     0.7907    0.7917    0.7906       600
weighted avg     0.7907    0.7917    0.7906       600
```

	snr_db	acc	bacc	f1_macro	auc_macro_ovr
0	0	0.705	0.705	0.704083	0.905200
1	10	0.825	0.825	0.821257	0.966267
2	18	0.845	0.845	0.846761	0.961533

РЕЗУЛЬТАТИ ЗОВНІШНЬОГО ТЕСТУВАННЯ (QIBOQUANTUMFINETUNED)



Висока якість розпізнавання класів BPSK та QAM16, для яких було отримано 143 правильні класифікації в обох випадках із 150. Основні похибки, як і у попередній моделі, зосереджені між QPSK та 8PSK полягали в тому, що клас QPSK у 53 випадках було помилково віднесено до 8PSK, а 8PSK у 43 випадках до QPSK.

ВИСНОВКИ

У межах роботи було не лише розглянуто теоретичні можливості Quantum Machine Learning, а й перевірено, як сучасні QML-фреймворки поводяться в реальній задачі класифікації модуляції радіосигналів. Найбільш перспективними за результатами порівняльного аналізу виявилися Qibo, TorchQuantum та Amazon Braket, оскільки вони поєднали достатньо високу якість класифікації, практичну придатність й різні підходи до організації квантово-класичних обчислень.

Реалізація задачі класифікації радіосигналів продемонструвала, що модель QiboFineTuned забезпечила високу точність та меншу обчислювальну вартість, тоді як TorchQuantumFineTuned продемонструвала стабільну end-to-end гібридну архітектуру та добру здатність до узагальнення на зовнішньому датасеті. Загалом результати підтвердили, що квантово-класичні моделі можуть бути використані для задач обробки радіосигналів, однак їх ефективність визначається не лише самою моделлю, а й усім експериментальним pipeline: підготовкою даних, вибором фреймворку, способом кодування ознак, стабільністю середовища та часом виконання.

ДЯКУЮ ЗА УВАГУ