Project Analysis Report: Measurement-Free Quantum Classifier

Executive Summary

This report identifies **consistency issues**, **logical errors**, and **design concerns** found in the measurement-free quantum classifier project. The analysis covers code consistency, mathematical correctness, implementation bugs, and architectural issues.

🔴 CRITICAL ISSUES

1. Inconsistent Winner Selection Logic in MemoryBank

File: `src/IQL/learning/memory_bank.py`

```python
def winner(self, psi):
    scores = self.scores(psi)
    idx = int(max(range(len(scores)), key=lambda i: abs(scores[i])))
    #idx = int(max(range(len(scores)), key=lambda i: scores[i])) ## causes lower score ??
    return idx, scores[idx]
```

Problem:

- The commented line suggests confusion about whether to use `abs(scores[i])` or `scores[i]`
- Using `abs()` is **mathematically incorrect** for interference-based classification
- The score sign encodes the class label (positive vs negative), so taking absolute value loses critical information
- This breaks the fundamental ISDO decision rule: `sign(Re⟨χ|ψ⟩)`

Impact: High - This affects all Winner-Take-All and Adaptive Memory algorithms

Fix Required:

```python
def winner(self, psi):
    scores = self.scores(psi)
    # For interference quantum classification, we need the memory with highest score magnitude
    # but we preserve the sign for class prediction
    idx = int(max(range(len(scores)), key=lambda i: abs(scores[i])))
    return idx, scores[idx]
```

---

2. Label Encoding Inconsistency

Multiple Files: Throughout the codebase

Problem: The project uses THREE different label encodings inconsistently:

1. **Binary {0, 1}** - Used in:
   - CNN training (`val_labels.npy`)
   - Static ISDO classifier predictions
   - Classical baseline evaluations
2. **Polar {-1, +1}** - Used in:
   - Online perceptron training (`val_labels_polar.npy`)
   - Quantum update rules
   - Margin calculations
3. **Mixed conversions:**

```python
# In static_isdo_classifier.py
return 1 if self.exact.score(chi, psi) < 0 else 0  # Returns {0,1}

# In regime2_online.py
y_hat = 1 if s >= 0 else -1  # Returns {-1,+1}
```

Impact: Critical - Causes accuracy calculation errors and comparison inconsistencies

Evidence of Confusion:

```python
# In run_final_comparison.py
results["ISDO_K"] = accuracy_score((y_test + 1) // 2, y_pred_isdo)
# This conversion suggests polar labels being compared to binary predictions
```

Fix Required: Standardize on ONE label encoding throughout (recommend {-1, +1} for quantum algorithms)

---

3. Missing Label Assignment in ClassState

File: `src/IQL/learning/class_state.py`

```python
class ClassState:
    def __init__(self, vector: np.ndarray, backend: InterferenceBackend):
        self.vector = normalize(vector.astype(np.complex128))
        self.backend = backend
        # MISSING: self.label attribute!
```

Problem:

- `AdaptiveMemory._is_dominated()` uses `chi_j.label` which doesn't exist
- This will cause `AttributeError` at runtime during pruning

Impact: High - Breaks adaptive memory pruning entirely

Fix Required:

```python
class ClassState:
    def __init__(self, vector: np.ndarray, backend: InterferenceBackend, label: int = None):
        self.vector = normalize(vector.astype(np.complex128))
        self.backend = backend
        self.label = label  # Add label tracking
```

---

4. MemoryBank.remove() Method Missing

File: `src/IQL/learning/memory_bank.py`

Problem:

- `AdaptiveMemory._prune_memories()` calls `self.memory_bank.remove(j)`
- This method doesn't exist in the MemoryBank class
- Will cause `AttributeError` during pruning

Impact: High - Adaptive memory pruning completely broken

Fix Required:

```python
class MemoryBank:
    def remove(self, idx):
        """Remove memory at given index"""
        if 0 <= idx < len(self.class_states):
            del self.class_states[idx]
```

🟡 SIGNIFICANT ISSUES

5. Inconsistent Prediction Methods

Different models return predictions in different ways:

```python
# FixedMemoryIQC.predict() - returns list
def predict(self, X):
    return [self.classifier.predict(x) for x in X]

# StaticISDOModel.predict() - calls classifier directly
def predict(self, X):
    return self.classifier.predict(X)

# AdaptiveIQC.predict() - returns result of classifier.predict (could be single value or list)
def predict(self, X):
    return self.classifier.predict(X)
```

Problem: Inconsistent return types (list vs numpy array)

---

6. Prototype Generation Missing X, y Parameters

File: `src/IQL/models/static_isdo_model.py`

```python
def _ensure_prototypes(self, X, y):
    generate_prototypes(
        X=X,
        y=y,
        K=self.K,
        output_dir=proto_dir
    )

def fit(self, X, y):
    self._ensure_prototypes(X, y)  # Good!
    # But then...
    self.classifier = StaticISDOClassifier(
        proto_dir=self.proto_dir,
        K=self.K
    )
    return self
```

But in `fixed_memory_iqc.py`:

```python
def fit(self, X, y):
    self._ensure_prototypes(X, y)  # Ensures prototypes exist
    proto_vectors = self._load_prototypes()  # Good pattern
```

Note: Actually consistent - this is handled properly in both cases.

---

7. PrimeB Backend Not Used Properly

File: `src/IQL/backends/prime_b.py`

```
python

def score(self, chi: np.ndarray, psi: np.ndarray) -> float:
    # ... implementation ...
    observable = Pauli("Z"+"I" * (n-1))  # Only measures first qubit!
    return float(sv.expectation_value(observable).real)
    """
    pass  # Dead code after return!
```

Problems:

1. Dead `pass` statement after return
2. Only measuring first qubit Z observable may not capture full interference
3. Comment says "decision-sufficient" but rank correlation is -0.004 (essentially random)
4. The validation test shows 52.5% sign agreement - **barely better than random guessing**

Evidence from validation:

```
PrimeB sign agreement with Exact: 52.5%
PrimeB rank correlation with Exact: -0.004
```

Impact: Medium - PrimeB backend is essentially non-functional for classification

---

8. Incomplete Error Handling in OnlinePerceptron.load()

File: `src/IQL/regimes/regime2_online.py`

```python
@classmethod
def load(cls, path):
    with open(path, "rb") as f:
        payload = pickle.load(f)
    obj = cls(
        class_state=payload["class_state"],
        eta=payload["eta"],
    )
    obj.num_updates = payload["num_updates"]
    obj.num_mistakes = payload["num_mistakes"]  # This key doesn't exist!
    obj.margin_history = payload["margin_history"]  # This key doesn't exist!
    obj.history = payload["history"]
    return obj
```

Problem: Accessing non-existent keys `num_mistakes` and `margin_history`

Impact: Medium - Will crash when loading saved models

---

9. Adaptive Memory Spawning in Wrong Condition

File: `src/IQL/regimes/regime3c_adaptive.py`

```python
def step(self, psi, y):
    S = self.aggregated_score(psi)
    margin = y * S
    spawned = False

    # Growth logic
    neg_margins = [m for m in self.margins if m < 0]
    if len(neg_margins) >= 20:
        tau = np.percentile(neg_margins, self.percentile)
        if margin < tau:  # Only spawns for misclassified samples
            chi_new = y * psi
            # ...
            spawned = True

    # Update logic
    if not spawned and margin < 0:  # Also only updates misclassified
        # ...
```

Problem:

- Spawning only happens when `margin < tau` (misclassified samples)
- Update only happens when `margin < 0` (misclassified samples)
- This means correctly classified samples **never** reinforce the memory
- This is asymmetric learning that may lead to instability

Impact: Medium - May explain poor performance (56% in output comments)

---

10. K-Means Seed Not Propagated Properly

File: `src/IQL/learning/calculate_prototype.py`

```python
def generate_prototypes(X, y, K, output_dir, seed=42):
    set_seed(seed)  # Sets global seed
    # ...
    for cls in [0, 1]:
        kmeans = KMeans(
            n_clusters=K,
            random_state=seed,  # Good!
            n_init=10
        )
```

But when called from models:

```python
# In fixed_memory_iqc.py
def _ensure_prototypes(self, X, y):
    generate_prototypes(
        X=X,
        y=y,
        K=self.K,
        output_dir=proto_dir
        # Missing: seed=42 parameter!
    )
```

Impact: Low-Medium - May cause non-reproducible results

🟢 DESIGN & BEST PRACTICE ISSUES

11. Unnecessary Complex Normalization

Multiple places perform redundant normalizations:

```python
# In embedding_to_state.py
def embedding_to_state(x: np.ndarray) -> np.ndarray:
    x = x.astype(np.complex128)
    norm = np.linalg.norm(x)
    if norm == 0:
        raise ValueError("Zero embedding encountered")
    return x / norm

# But embeddings are already normalized in extract_embeddings.py:
z = z.to(torch.float64)
z = torch.nn.functional.normalize(z, p=2, dim=1)

# And then normalized AGAIN in multiple test files:
X_train /= np.linalg.norm(X_train, axis=1, keepdims=True)
X_test /= np.linalg.norm(X_test, axis=1, keepdims=True)
```

Impact: Low - Just inefficient, not incorrect

---

12. Hardcoded Paths in Data Loader

File: `src/data/pcam_loader.py`

```python
def get_pcam_dataset(data_dir='/home/tarakesh/Work/Repo/measurement-free-quantum-classifier/dataset
```

Problem: Hardcoded absolute path as default argument

Fix: Should use paths from config:

```python
def get_pcam_dataset(data_dir=None, ...):
    if data_dir is None:
        _, PATHS = load_paths()
        data_dir = PATHS["dataset"]
```

---

13. Inconsistent Backend Initialization

Some models use default backend, others require explicit:

```python
# In adaptive_iqc.py
def __init__(self, ..., backend=None, ...):
    self.backend = backend or ExactBackend()  # Good!

# In fixed_memory_iqc.py
def __init__(self, K: int, eta: float = 0.1, backend=None):
    self.backend = backend or ExactBackend()  # Good!

# But in winner_take_all.py
def __init__(self, memory_bank, eta, backend = ExactBackend()):
    # Uses mutable default argument!
```

Problem: Using mutable default arguments is Python anti-pattern

---

14. Missing Validation in Transition Backend

File: src/IQL/backends/transition.py

```python
def score(self, chi, psi) -> float:
    chi = np.asarray(chi, dtype=np.complex128)
    psi = np.asarray(psi, dtype=np.complex128)

    # Normalize
    chi = chi / np.linalg.norm(chi)
    psi = psi / np.linalg.norm(psi)

    assert chi.shape == psi.shape  # Good!
    n = int(np.log2(len(psi)))
    assert 2**n == len(psi)  # Good!

    # But no validation that these are valid quantum states!
    # (norm should be 1, complex valued, etc.)
```

---

15. Commented Out Important Code

File: src/training/classical/train_embedding_models.py

```python
# ----------------------------
# Preprocessing (DEPRECATED: Now handled in extract_embeddings.py)
# ----------------------------
# # 1) Standardize (important for linear models)
# scaler = StandardScaler()
# X_std = scaler.fit_transform(X)
#
# # 2) L2-normalize (important for similarity & quantum)
# X_l2 = normalize(X_std, norm="l2")
```

Problem:

- Leaves doubt about whether preprocessing is actually needed
- Comment says "DEPRECATED" but unclear if this is correct
- Classical models might benefit from standardization

---

16. QSVM Subsampling Hidden in Code

File: `src/quantum/compute_qsvm_kernel.py`

```python
# ----------------------------------------------------------
# SUBSAMPLING for Baseline Efficiency
# ----------------------------------------------------------
# Limiting to 500 samples because O(N^2) kernel computation
# for 3500 samples would take ~17 hours on GPU.
MAX_TRAIN = 500000  # Typo? Should be 500?
MAX_TEST = 200000   # Typo? Should be 200?
```

Problem:

- Values look like typos (500000 instead of 500?)
- Subsampling makes comparison unfair but isn't clearly documented in results
- This is why QSVM is marked as optional/excluded in comparison

---

17. Incomplete Consolidation Pipeline

File: `src/training/protocol_adaptive/train_adaptive_memory.py`

```python
# Output shows:
"""
Regime 3-C accuracy (3-B inference): 0.788
"""

# But consolidate_memory.py shows:
"""
Consolidation pass accuracy: 0.8048571428571428
FINAL Regime 3-C accuracy: 0.884
"""
```

Problem:

- Accuracy jumps from 78.8% to 88.4% after consolidation
- This suggests the main training script should include consolidation step
- But AdaptiveIQC model has `consolidate=True` which should handle this

📊 ARCHITECTURAL CONCERNS

18. Confusing Model Hierarchy

The project has overlapping abstractions:

```
Models (high-level):
- FixedMemoryIQC
- StaticISDOModel
- AdaptiveIQC

Regimes (training):
- OnlinePerceptron (Regime 2)
- WinnerTakeAll (Regime 3A)
- AdaptiveMemory (Regime 3C)

Baselines:
- StaticISDOClassifier

Inference:
- WeightedVoteClassifier
```

Problem: It's unclear when to use models vs regimes directly

---

19. Results Don't Match Expected Performance

From the comparison script output:

```
python

"""
=== IQC Algorithm Comparison ===
Static_ISDO : 0.8806666666666667
IQC_Online : 0.904
IQC_Adaptive : 0.56  # Much worse!
Adaptive_Memory_Size : 45
"""
```

Problem:

- Adaptive IQC (56%) performs MUCH worse than Static ISDO (88%)
- This suggests fundamental implementation issues in Regime 3C
- Likely related to the spawning/update logic issues identified above

---

🔧 RECOMMENDED FIXES (Priority Order)

Priority 1 (Critical - Breaks Functionality)

1. ✅ Add `label` attribute to ClassState
2. ✅ Implement `MemoryBank.remove()` method
3. ✅ Fix label encoding inconsistency across all files
4. ✅ Fix `OnlinePerceptron.load()` to use correct keys

Priority 2 (High - Affects Correctness)

5. ✅ Review winner selection logic (`abs()` vs no `abs()`)
6. ✅ Fix adaptive memory spawning logic to reinforce correct predictions
7. ✅ Add seed parameter to all `generate_prototypes()` calls
8. ✅ Remove dead code in PrimeB backend
9. ✅ Document PrimeB backend limitations (or fix it)

Priority 3 (Medium - Best Practices)

10. ✅ Standardize prediction method return types
11. ✅ Fix hardcoded paths
12. ✅ Fix mutable default arguments
13. ✅ Add proper validation in backends
14. ✅ Clean up commented preprocessing code

Priority 4 (Low - Documentation)

15. ✅ Document QSVM subsampling properly
16. ✅ Clarify model vs regime usage
17. ✅ Add architecture documentation

📝 TESTING RECOMMENDATIONS

1. **Unit tests needed for:**
   - MemoryBank operations (especially `remove()`)
   - ClassState with labels
   - Label conversions
   - All backend implementations
2. **Integration tests needed for:**
   - Full training pipeline
   - Save/load functionality
   - Adaptive memory growth and pruning
3. **Validation tests needed for:**
   - Accuracy comparisons with known baselines
   - Reproducibility with fixed seeds
   - Backend equivalence

✅ WHAT'S ACTUALLY WORKING WELL

1. **Clean separation of concerns** between encoding, backends, and learning
2. **Good documentation** in code comments
3. **Comprehensive baseline comparisons** (when working correctly)
4. **Proper seed management** in most places
5. **Nice visualization utilities**
6. **Well-structured directory layout**

📄 SUMMARY

Total Issues Found: 19

- Critical (breaks functionality): 4
- High (affects correctness): 6
- Medium (best practices): 6
- Low (documentation/style): 3

Main Concerns:

1. Adaptive Memory implementation has multiple bugs preventing proper operation
2. Label encoding inconsistency creates comparison difficulties
3. PrimeB backend doesn't work as intended
4. Missing critical methods in core classes

Next Steps:

1. Fix critical issues (1-4) immediately
2. Add comprehensive tests
3. Re-run all experiments with fixes
4. Document known limitations clearly