

QCardEst/QCardCorr: Key Computation Functions Guide

QCardEst/QCardCorr

January 11, 2026

The difference between QCardEst and QCardCorr is controlled by the `valueType` setting:

- **QCardEst:** `valueType = "rows"` → Direct cardinality prediction
- **QCardCorr:** `valueType = "rowFactor"` → Correction factor prediction (multiplies existing prediction)

1 Entry Point: Main Execution Flow

File: `runRegression.py`

Key Function: Main execution

```
# Lines 49-65: Setup and execution
env = CardEnv(inputFile=settings["data"] + ".csv", settings=settings)
agent = interpretation.fromString(settings["loss"], None, env)
model = vqc(settings=settings, nInputs=env.getInputSize(), nOutputs=agent.nInputs, no
optimizer = Regression(agent, env, settings)
optimizer.run() # Main training loop
optimizer.listSolutions() # Generate final predictions
```

Key Input: `settings["valueType"]` determines Est vs Corr mode

2 Training Loop: Core Learning Process

File: `ML/GradientQML.py`

Key Function: `GradientQML.run()` (lines 43-80)

Critical Computation Steps:

```
# Line 55: Get training sample
state, expected = self.env.step()

# Line 56: Forward pass through quantum model
prediction = self.agent(Tensor(state))
```

```

# Line 57: Compute loss
cross, logState = self.interpret(state, expected, prediction, logState)

# Lines 63-65: Backward pass and optimization
self.optimizer.zero_grad()
loss.backward()
self.optimizer.step()

# Lines 72-74: Periodic evaluation and logging
stats = self.env.elvaluateModel(self.agent)
self.resultFile.write(",".join(str(e) for e in ([episode] + stats)))

```

Key Input:

- state: Feature vector (query tables + selectivities)
- expected: Target value (computed based on valueType)

3 Expected Value Computation: Est vs Corr Difference

File: cardEnv.py

Key Function: CardEnv.expectedWithType() (lines 34-39)

Critical Logic:

```

def expectedWithType(self, entry, type):
    if type in ["rowFactor", "rowsFactor"]: # QCardCorr
        return self.expectedMapping(entry["rows"] / entry["rowsPredicted"])
    if type in ["costFactor", "costsFactor"]:
        return self.expectedMapping(entry["cost"] / entry["costPredicted"])
    return self.expectedMapping(entry[type]) # QCardEst: direct value

def expectedMapping(self, value):
    return math.log(value) # Log-space transformation

```

Key Difference:

- QCardEst ("rows"): expected = $\log(\text{true_cardinality})$
- QCardCorr ("rowFactor"): expected = $\log(\text{true_cardinality} / \text{predicted_cardinality})$

4 Model Evaluation: Metrics Computation

File: cardEnv.py

Key Function: CardEnv.elvaluateModel() (lines 87-139)

Critical Computation (lines 108-135):

```

for i, prediction in enumerate(predictions):
    entry = evaluationSet[i]

```

```

value = model.predict(prediction).item() # Model output
expected = self.expected(entry) # True value

# Compute error metrics
diffs.append(abs(value - expected))
factors.append(compute_factor(value, expected))

# Convert to cardinality space
if "Factor" in self.valueType: # QCardCorr
    correction = math.exp(value)
    card = math.log(correction * entry["rowsPredicted"])
else: # QCardEst
    card = value

expectedCard = math.log(entry["rows"])
diffCards.append(abs(card - expectedCard))
factorCards.append(compute_factor(card, expectedCard))

# Return statistics: [mean_diff, median_diff, var_diff, mean_factor, ...]
return [stat.mean(diffs), stat.median(diffs), ...]

```

Key Output: 13-element statistics array written to CSV (line 73 in GradientQML.py)

5 Final Predictions: Solution Generation

File: cardEnv.py

Key Function: CardEnv.listSolutions() (lines 141-155)

Critical Computation:

```

for index, entry in enumerate(self.data):
    state = Tensor(entry["features"])
    temp = model(state) # Quantum model output
    prediction = model.predict(temp) # Post-processed prediction
    expected = self.expected(entry) # True value
    loss = model.loss(temp, expected) # Loss value

    # Compute factor (error metric)
    factor = abs(prediction / expected)
    if factor < 1 and factor != 0:
        factor = 1 / factor

    result.append([index, prediction.item(), expected, factor.item(), loss.item()])

```

Key Output: CSV file with columns: id, prediction, expected, factor, loss

6 Quantum Model Architecture

File: ML/models.py

Key Function: vqc() (lines 31-87)

Critical Components:

```
# Lines 44-49: Create parameterized quantum circuit
qc = circuits.parameterizedCircuit(nQubits, settings)
X = list(qc.parameters)[:divider] # Input parameters
params = list(qc.parameters)[divider:] # Trainable parameters

# Lines 63-67: Connect quantum circuit to PyTorch
qnn = CircuitQNN(qc, input_params=X, weight_params=params, quantum_instance=qi)
quantumNN = TorchConnector(qnn, initialWeights)

# Lines 70-85: Build complete model
model = torch.nn.Sequential(
    paddingLayer,          # Pad input to qubit count
    quantumNN,             # Quantum neural network
    ReshapeSumLayer,       # Reduce quantum states to outputs
    NormLayer              # Normalize probabilities
)
```

Key Input: settings dict with:

- encoding: ["rx", "rz"] (feature encoding)
- reps: Number of layers
- calc: "yz" (measurement basis)
- entangleType: "circular" (entanglement pattern)

7 Post-Processing: Interpretation Layers

File: ML/regressionInterpretation.py

Key Function: fromString() (lines 244-274)

Available Interpretations:

- "linear": LinearScale - Single probability with scaling
- "rational": Rational - Ratio of two probabilities
- "rationalLog": RationalLog - Log of ratio
- "threshold": SecondValueThreshold - Threshold-controlled
- "PlaceValue": PlaceValueSystem - Place-value encoding

Key Function: Interpretation.predict() (line 32)

```
def predict(self, prediction: torch.Tensor):
    """Convert quantum output to scalar prediction"""
    return prediction
```

Key Function: Interpretation.loss() (line 38)

```
def loss(self, prediction: torch.Tensor, compare):
    """MSE loss for optimization"""
    return (self.predict(prediction) - compare)**2
```

8 Data Loading and Feature Extraction

File: cardEnv.py

Key Function: CardEnv.load_data() (lines 47-78)

Data Format (from costs/*.csv):

1. Tablenames (separated by ';')
2. Selectivities (separated by ';')
3. PostgreSQL cost
4. PostgreSQL/MSCN cardinality prediction
5. Actual execution time
6. True cardinality

Key Processing:

```
# Lines 56-67: Extract features
query = tmp[0].split(";")
selectivities = tmp[1].split(";")
features = [[table_index, selectivity] for table, sel in zip(query, selectivities)]

# Line 76: Create data entry
datapoint = {
    "id": id,
    "features_raw": features,
    "features": self.feature_map(features), # Map to [0, ]
    "rows": int(values[3]), # True cardinality
    "rowsPredicted": int(values[1]), # Classical prediction
    "cost": float(values[2]),
    "costPredicted": float(values[0])
}
```

9 Result File Format

Training Logs: results/*.csv

Format: episode, mean_diff, median_diff, var_diff, mean_factor, median_factor, var_factor, mean_diffCards, median_diffCards, var_diffCards, mean_factorCards, median_factorCards, var_factorCards, close

Generated by: GradientQML.run() line 73

Solutions: results/solutions/*.sl.csv

Format: id, prediction, expected, factor, loss

Generated by: CardEnv.listSolutions() line 154

10 Summary: Key Files for Results Computation

1. runRegression.py: Entry point, settings configuration
2. ML/GradientQML.py: Training loop, optimization, result logging
3. cardEnv.py:
 - expectedWithType(): Est vs Corr difference
 - elvaluateModel(): Metrics computation
 - listSolutions(): Final predictions
4. ML/models.py: Quantum model architecture (vqc())
5. ML/regressionInterpretation.py: Post-processing layers
6. cardEnv.py (load_data): Data loading and feature extraction

11 Key Differences: QCardEst vs QCardCorr

Aspect	QCardEst (valueType="rows")	QCardCorr (valueType="rowFactor")
Expected Value	$\log(\text{true_cardinality})$	$\log(\text{true_cardinality} / \text{predicted_cardinality})$
Model Output	Direct cardinality in log space	Correction factor in log space
Final Cardinality	$\exp(\text{prediction})$	$\exp(\text{prediction}) * \text{predicted_cardinality}$
Use Case	Predict from scratch	Correct existing predictions