In [1]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

In [2]:
```
# Load dataset and preprocess
iris = load_iris()
X, y = iris.data, iris.target
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [3]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randc
```

In [4]:
```
# Antibody class (multi-class)
class Antibody:
    def __init__(self, n_features=4, n_classes=3):
        self.weights = np.random.randn(n_classes, n_features)
        self.bias = np.random.randn(n_classes)
        self.affinity = 0.0

    def predict(self, x):
        logits = np.dot(self.weights, x) + self.bias
        return np.argmax(logits)

    def evaluate(self, X, y, reg_lambda=0.01):
        preds = [self.predict(x) for x in X]
        acc = accuracy_score(y, preds)
        reg_term = reg_lambda * np.sum(self.weights**2)  # L2 regularization
        self.affinity = acc - reg_term  # penalized affinity
        return self.affinity
```

In [5]:
```
# CLONALG with regularization + early stopping
def clonal_selection(
    X, y, pop_size=20, generations=50, clones_per=5,
    mutation_rate=0.1, reg_lambda=0.01, patience=5
):
    population = [Antibody() for _ in range(pop_size)]
    best_affinity = -np.inf
    patience_counter = 0

    for gen in range(generations):
        for ab in population:
            ab.evaluate(X, y, reg_lambda)

        population.sort(key=lambda ab: ab.affinity, reverse=True)
        best = population[:pop_size // 2]

        if best[0].affinity > best_affinity + 1e-4:
            best_affinity = best[0].affinity
            patience_counter = 0
        else:
            patience_counter += 1

        print(f"Generation {gen+1} - Best Affinity: {best_affinity:.4f}")
        if patience_counter >= patience:
            print("Early stopping due to no improvement.")
            break

        # Cloning and mutation
        clones = []
        for ab in best:
            for _ in range(clones_per):
                clone = Antibody()
                clone.weights = ab.weights + np.random.normal(0, mutation_rate
                clone.bias = ab.bias + np.random.normal(0, mutation_rate, ab.b
                clones.append(clone)

        for clone in clones:
            clone.evaluate(X, y, reg_lambda)

        population = sorted(best + clones, key=lambda ab: ab.affinity, reverse

    return population[0]
```

In [6]:
```
# Train the model
best_model = clonal_selection(X_train, y_train)
```

```
Generation 1 - Best Affinity: 0.6473
Generation 2 - Best Affinity: 0.6740
Generation 3 - Best Affinity: 0.7491
Generation 4 - Best Affinity: 0.7784
Generation 5 - Best Affinity: 0.7802
Generation 6 - Best Affinity: 0.8266
Generation 7 - Best Affinity: 0.8424
Generation 8 - Best Affinity: 0.8603
Generation 9 - Best Affinity: 0.8745
Generation 10 - Best Affinity: 0.8887
Generation 11 - Best Affinity: 0.8896
Generation 12 - Best Affinity: 0.8935
Generation 13 - Best Affinity: 0.9069
Generation 14 - Best Affinity: 0.9127
Generation 15 - Best Affinity: 0.9127
Generation 16 - Best Affinity: 0.9246
Generation 17 - Best Affinity: 0.9246
Generation 18 - Best Affinity: 0.9247
Generation 19 - Best Affinity: 0.9342
Generation 20 - Best Affinity: 0.9342
Generation 21 - Best Affinity: 0.9433
Generation 22 - Best Affinity: 0.9433
Generation 23 - Best Affinity: 0.9433
Generation 24 - Best Affinity: 0.9504
Generation 25 - Best Affinity: 0.9520
Generation 26 - Best Affinity: 0.9520
Generation 27 - Best Affinity: 0.9520
Generation 28 - Best Affinity: 0.9541
Generation 29 - Best Affinity: 0.9541
Generation 30 - Best Affinity: 0.9541
Generation 31 - Best Affinity: 0.9644
Generation 32 - Best Affinity: 0.9644
Generation 33 - Best Affinity: 0.9644
Generation 34 - Best Affinity: 0.9644
Generation 35 - Best Affinity: 0.9644
Generation 36 - Best Affinity: 0.9644
Early stopping due to no improvement.
```

In [7]:
```
# Evaluate on test data
preds = [best_model.predict(x) for x in X_test]
acc = accuracy_score(y_test, preds)
print(f"\nFinal Test Accuracy: {acc * 100:.2f}%")
```

```
Final Test Accuracy: 95.56%
```

In [ ]:
```
```