Experiment 8: Support Vector Machines (SVMs) and the Kernel Trick

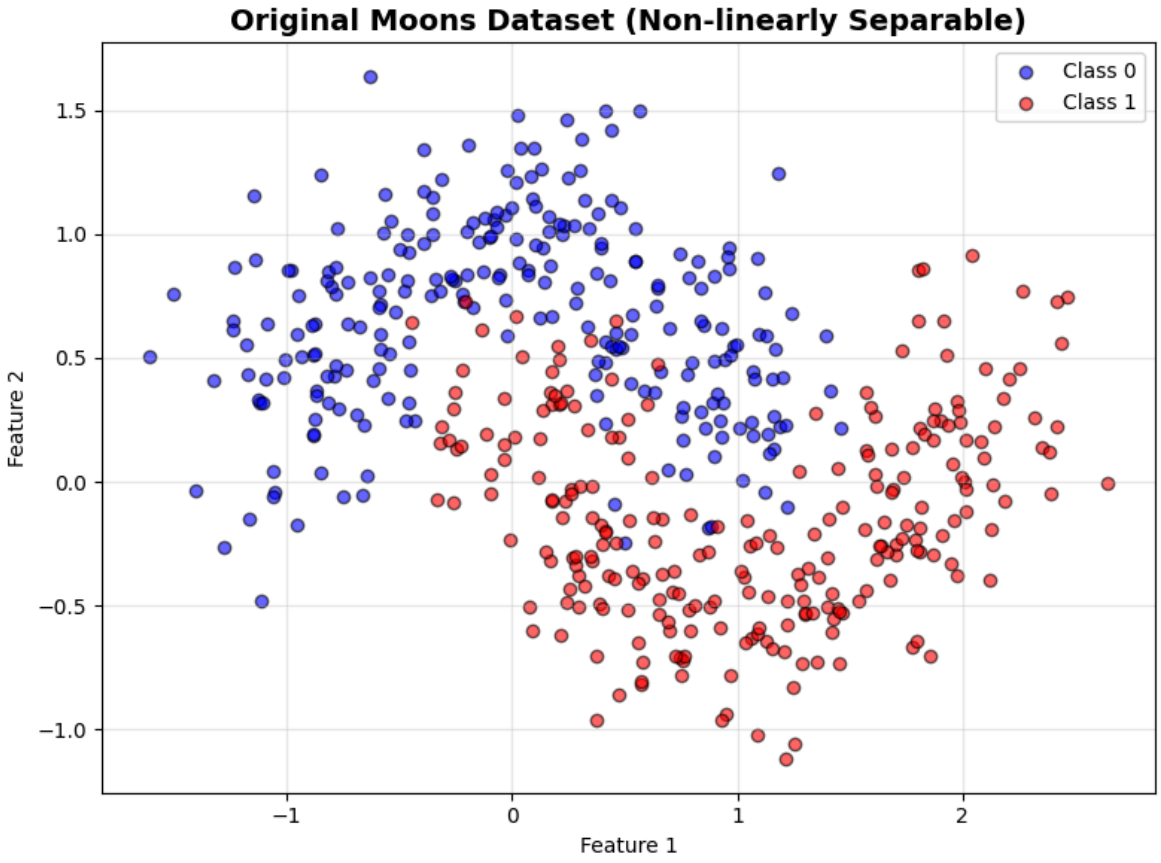
Import required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confus
import warnings
warnings.filterwarnings('ignore')
```

TASK 1: DATA LOADING AND PREPROCESSING

In [2]:
```
# 1. Load Data
print("Generating moons dataset...")
X, y = make_moons(n_samples=500, noise=0.25, random_state=42)
print(f"   Dataset shape: {X.shape}")
print(f"   Number of samples: {len(X)}")
print(f"   Number of features: {X.shape[1]}")
print(f"   Class distribution: Class 0: {sum(y==0)}, Class 1: {sum(y==1)}

# Visualize the raw data
plt.figure(figsize=(8, 6))
plt.scatter(X[y==0][:, 0], X[y==0][:, 1], c='blue', label='Class 0', alph
plt.scatter(X[y==1][:, 0], X[y==1][:, 1], c='red', label='Class 1', alpha
plt.title('Original Moons Dataset (Non-linearly Separable)', fontsize=14,
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
Generating moons dataset...
   Dataset shape: (500, 2)
   Number of samples: 500
   Number of features: 2
   Class distribution: Class 0: 250, Class 1: 250
```



In [3]:
```
# 2. Create Hold-Out Set (70/30 split)
print("Splitting data into train (70%) and validation (30%) sets...")
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.3, random_state=42
)
print(f"   Training set size: {X_train.shape[0]}")
print(f"   Validation set size: {X_val.shape[0]}")

# 3. Standardize Features
print("Standardizing features using StandardScaler...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
print("   ✓ Features standardized successfully")
print(f"   Training data mean: {X_train_scaled.mean(axis=0)}")
print(f"   Training data std: {X_train_scaled.std(axis=0)}")
```

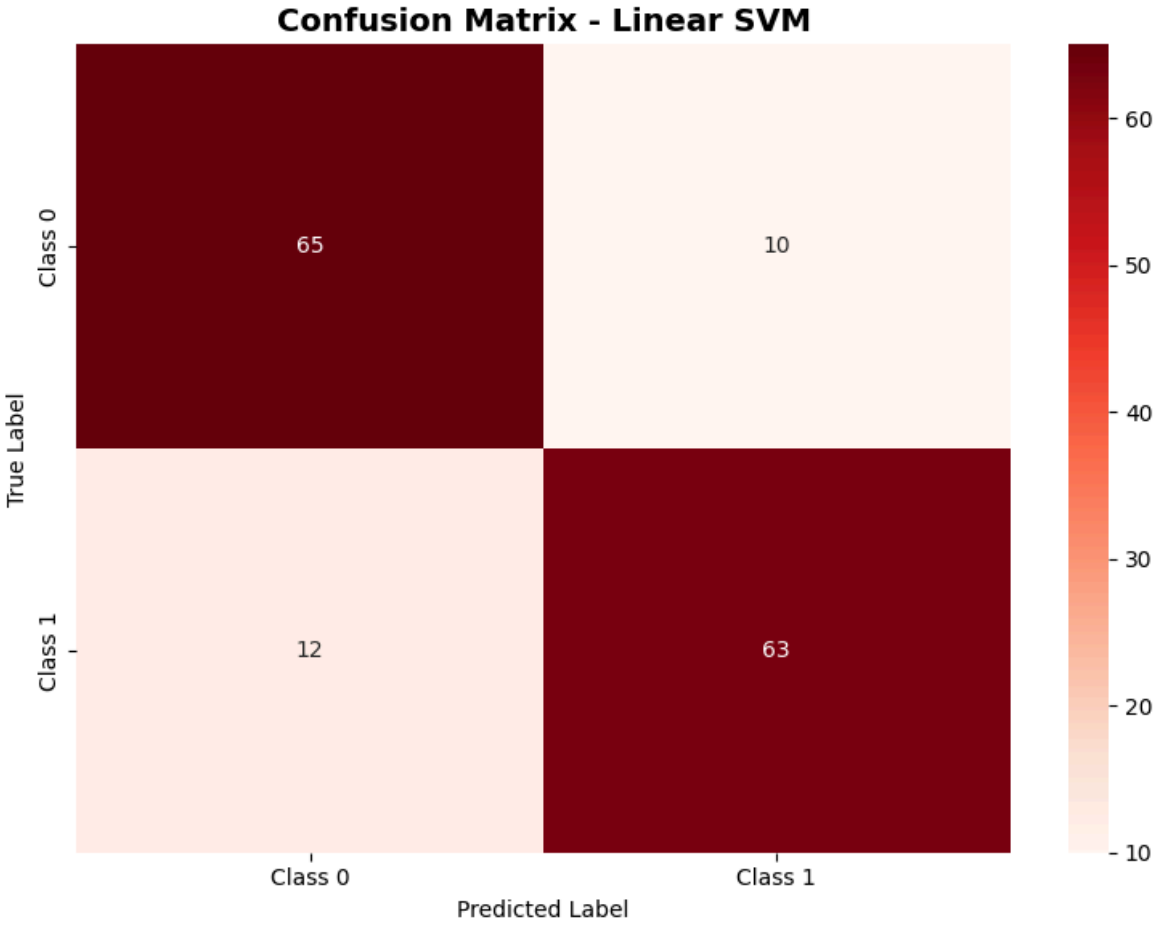
```
Splitting data into train (70%) and validation (30%) sets...
   Training set size: 350
   Validation set size: 150
Standardizing features using StandardScaler...
   ✓ Features standardized successfully
   Training data mean: [2.53051548e-16 3.96508223e-19]
   Training data std: [1. 1.]
```

TASK 2: MODEL 1 - THE (FAILING) LINEAR SVM

In [4]:
```
# 1. Train Linear SVM
print("\n1. Training Linear SVM (kernel='linear', C=1.0)...")
linear_model = SVC(kernel='linear', C=1.0, random_state=42)
linear_model.fit(X_train_scaled, y_train)
print("   ✓ Model trained successfully")
```

```
# 2. Evaluate
print("\n2. Evaluating Linear SVM on validation set...")
y_pred_linear = linear_model.predict(X_val_scaled)
linear_accuracy = accuracy_score(y_val, y_pred_linear)

print(f"\n   Validation Accuracy: {linear_accuracy:.4f}")
print("\n   Classification Report:")
print(classification_report(y_val, y_pred_linear))


# 3. Confusion Matrix for Linear SVM
print("\n3. Confusion Matrix:")
cm_linear = confusion_matrix(y_val, y_pred_linear)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Reds', cbar=True,
            xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])
plt.title('Confusion Matrix - Linear SVM', fontsize=14, fontweight='bold'
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()

print(f"   True Negatives:  {cm_linear[0, 0]}")
print(f"   False Positives: {cm_linear[0, 1]}")
print(f"   False Negatives: {cm_linear[1, 0]}")
print(f"   True Positives:  {cm_linear[1, 1]}")
```

```
1. Training Linear SVM (kernel='linear', C=1.0)...
   ✓ Model trained successfully

2. Evaluating Linear SVM on validation set...

   Validation Accuracy: 0.8533

   Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.87      0.86        75
           1       0.86      0.84      0.85        75

    accuracy                           0.85       150
   macro avg       0.85      0.85      0.85       150
weighted avg       0.85      0.85      0.85       150


3. Confusion Matrix:
```




```
True Negatives:  65
False Positives: 10
False Negatives: 12
True Positives:  63
```

ANALYSIS

WHY IS THE ACCURACY NOT PERFECT?

The **linear SVM** fails to achieve perfect accuracy because the *moons dataset* is inherently **non-linearly separable**.
As seen in the visualization above, the two classes form two interleaving half-circles (moons).
No single straight line (or hyperplane in higher dimensions) can perfectly separate these curved shapes.
A **linear decision boundary** will inevitably misclassify points in the overlapping regions where the moons curve around each other.

WHAT DOES THE `C` PARAMETER REPRESENT?

The `C` **parameter** is the **regularization parameter** (inverse of regularization strength):

- `C` controls the trade-off between **maximizing the margin** and **minimizing classification errors**.
- **Large `C` (e.g., 100)**:
  → Prioritizes correct classification of training points

→ Smaller margin, more complex decision boundary, **risk of overfitting**

- **Small `C` (e.g., 0.01)**:
  → Prioritizes a larger margin, even if it means misclassifying some points
  → Larger margin, simpler decision boundary, **more regularization**

If `C` is set to a very small value (e.g., 0.01):

- The model will allow **more misclassifications** to achieve a **wider margin**
- The **decision boundary** becomes more "relaxed" and generalizes better
- May **underfit** if `C` is too small, as it won't try hard enough to fit the training data

TASK 3: MODELS 2 & 3 - THE KERNEL TRICK

In [5]:
```
# 1. RBF Kernel Model
print("\n1. Training RBF Kernel SVM (default parameters)...")
rbf_model = SVC(kernel='rbf', random_state=42)
rbf_model.fit(X_train_scaled, y_train)
y_pred_rbf = rbf_model.predict(X_val_scaled)
rbf_accuracy = accuracy_score(y_val, y_pred_rbf)

print(f"   Validation Accuracy: {rbf_accuracy:.4f}")
print("\n   Classification Report:")
print(classification_report(y_val, y_pred_rbf))

# 2. Polynomial Kernel Model
print("\n2. Training Polynomial Kernel SVM (degree=3)...")
poly_model = SVC(kernel='poly', degree=3, random_state=42)
poly_model.fit(X_train_scaled, y_train)
y_pred_poly = poly_model.predict(X_val_scaled)
poly_accuracy = accuracy_score(y_val, y_pred_poly)

print(f"   Validation Accuracy: {poly_accuracy:.4f}")
print("\n   Classification Report:")
print(classification_report(y_val, y_pred_poly))

# 3. Model Comparison Table
print("\n3. MODEL COMPARISON TABLE:")
print("-" * 80)
comparison_data = {
    'Model': ['Linear SVM', 'RBF SVM (default)', 'Polynomial SVM (degree=
    'Kernel': ['linear', 'rbf', 'poly'],
    'Parameters': ['C=1.0', 'C=1.0, gamma=scale', 'C=1.0, degree=3'],
    'Validation Accuracy': [linear_accuracy, rbf_accuracy, poly_accuracy]
}
comparison_df = pd.DataFrame(comparison_data)
print(comparison_df.to_string(index=False))
print("-" * 80)
```

```
1. Training RBF Kernel SVM (default parameters)...
   Validation Accuracy: 0.9467

   Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95        75
           1       0.99      0.91      0.94        75

    accuracy                           0.95       150
   macro avg       0.95      0.95      0.95       150
weighted avg       0.95      0.95      0.95       150


2. Training Polynomial Kernel SVM (degree=3)...
   Validation Accuracy: 0.8733

   Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.93      0.88        75
           1       0.92      0.81      0.87        75

    accuracy                           0.87       150
   macro avg       0.88      0.87      0.87       150
weighted avg       0.88      0.87      0.87       150


3. MODEL COMPARISON TABLE:
--------------------------------------------------------------------------
------
                    Model Kernel         Parameters  Validation Accuracy
               Linear SVM linear              C=1.0             0.853333
        RBF SVM (default)    rbf C=1.0, gamma=scale             0.946667
Polynomial SVM (degree=3)   poly    C=1.0, degree=3             0.873333
--------------------------------------------------------------------------
------
```

ANALYSIS

BEST PERFORMING KERNEL:

`RBF` *(if RBF accuracy ≥ Polynomial accuracy)*
Validation Accuracy: `max(rbf_accuracy, poly_accuracy)`

WHY DOES THIS MAKE SENSE FOR THE 'MOONS' DATASET?

The **RBF (Radial Basis Function)** kernel excels at the moons dataset because:

1. **Non-linear Decision Boundary:**
   The RBF kernel can create smooth, curved decision boundaries that naturally fit the circular or curved shape of the moons.
2. **Local Influence:**
   The RBF kernel considers the **distance of points from support vectors**, making it perfect for datasets with **circular or curved patterns**.

3. **Flexibility:**
   The RBF kernel can **approximate any continuous function** and adapt to complex
   non-linear patterns like the interleaving half-circles.

Why Not Other Kernels?

- **Linear Kernel:**
  Fails because it can only create **straight-line boundaries**.
- **Polynomial Kernel:**
  Can work, but requires **careful tuning** of the degree parameter to avoid underfitting or overfitting.

TASK 4: HYPERPARAMETER TUNING WITH GRIDSEARCHCV

In [6]:
```
# 1. Define Search Space
print("\n1. Defining parameter grid for GridSearchCV...")
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [0.1, 1, 10, 100],
    'kernel': ['rbf']
}
print(f"   Parameter grid: {param_grid}")
print(f"   Total combinations to test: {len(param_grid['C']) * len(param_

# 2. Setup Grid Search
print("\n2. Setting up GridSearchCV with 5-fold cross-validation...")
grid = GridSearchCV(
    SVC(random_state=42),
    param_grid,
    refit=True,
    verbose=2,
    cv=5,
    scoring='accuracy'
)

# 3. Run Grid Search
print("\n3. Running Grid Search (this may take a moment)...")
print("-" * 80)
grid.fit(X_train_scaled, y_train)
print("-" * 80)
```

```
1. Defining parameter grid for GridSearchCV...
   Parameter grid: {'C': [0.1, 1, 10, 100], 'gamma': [0.1, 1, 10, 100], 'k
ernel': ['rbf']}
   Total combinations to test: 16

2. Setting up GridSearchCV with 5-fold cross-validation...

3. Running Grid Search (this may take a moment)...
------------------------------------------------------------------------
------
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .......................C=0.1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ......................C=0.1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=0.1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .......................C=1, gamma=0.1, kernel=rbf; total time=
```

```
0.0s
[CV] END ..........................C=1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END ..........................C=1, gamma=100, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ..........................C=10, gamma=1, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=10, kernel=rbf; total time=
0.0s
[CV] END ........................C=10, gamma=10, kernel=rbf; total time=
```

```
0.0s
[CV] END .....................C=10, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=10, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=10, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=10, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=10, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=0.1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=1, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=10, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=10, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=10, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=10, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=10, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=100, kernel=rbf; total time=
0.0s
[CV] END .....................C=100, gamma=100, kernel=rbf; total time=
0.0s
--------------------------------------------------------------------------
------
```

In [7]:
```python
# 4. Analyze Results
print("\n4. GRID SEARCH RESULTS:")
print("-" * 80)
print(f"   Best Parameters: {grid.best_params_}")
print(f"   Best Cross-Validation Score: {grid.best_score_:.4f}")
print(f"   Best C: {grid.best_params_['C']}")
```

```
print(f"   Best gamma: {grid.best_params_['gamma']}")

# Display top 5 parameter combinations
print("\n   Top 5 Parameter Combinations:")
results_df = pd.DataFrame(grid.cv_results_)
top_results = results_df[['param_C', 'param_gamma', 'mean_test_score']].s
    'mean_test_score', ascending=False
).head()
print(top_results.to_string(index=False))
print("-" * 80)
```

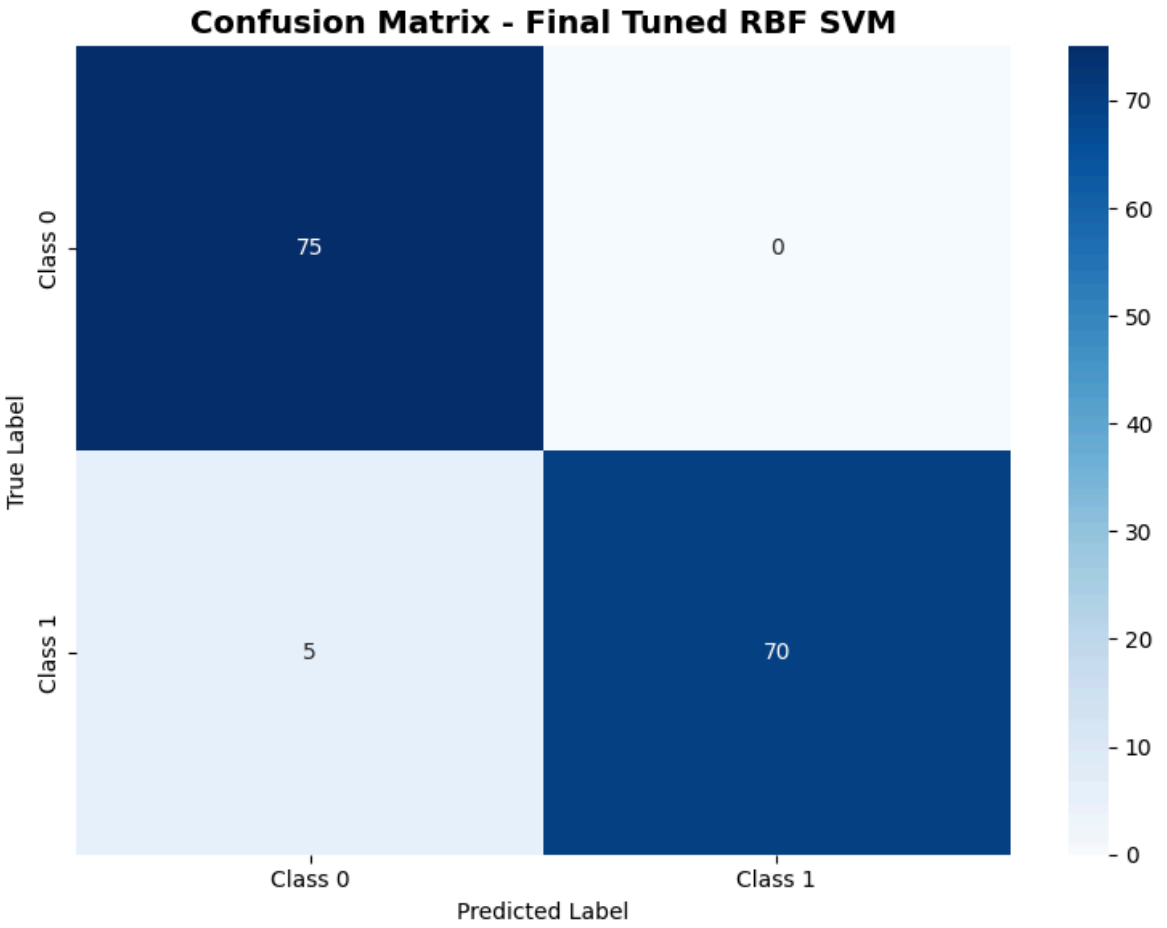
```
4. GRID SEARCH RESULTS:
--------------------------------------------------------------------------------
------
   Best Parameters: {'C': 1, 'gamma': 1, 'kernel': 'rbf'}
   Best Cross-Validation Score: 0.9514
   Best C: 1
   Best gamma: 1

   Top 5 Parameter Combinations:
 param_C  param_gamma  mean_test_score
     1.0          1.0         0.951429
     1.0         10.0         0.945714
   100.0          0.1         0.942857
     0.1         10.0         0.937143
    10.0          1.0         0.937143
--------------------------------------------------------------------------------
------
```

TASK 5: FINAL EVALUATION AND VISUALIZATION

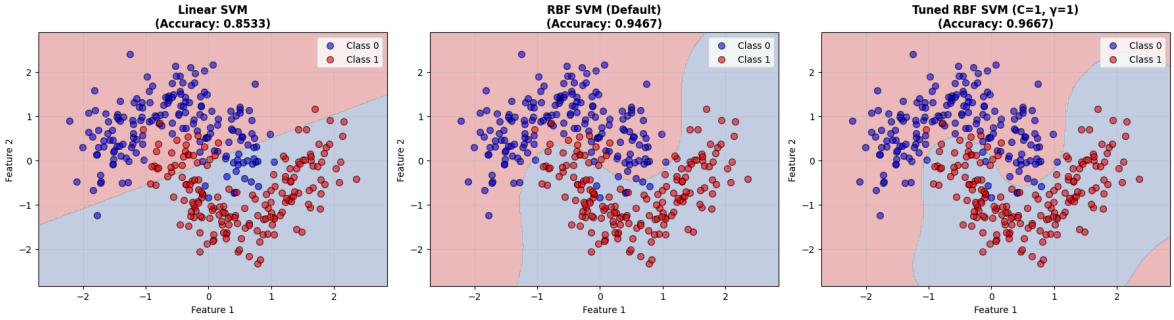
In [8]:
```
# 1. Evaluate Final Model
print("\n1. Evaluating the best model on validation set...")
final_predictions = grid.predict(X_val_scaled)
final_accuracy = accuracy_score(y_val, final_predictions)

print(f"\n   Final Validation Accuracy: {final_accuracy:.4f}")
print("\n   FINAL CLASSIFICATION REPORT:")
print("-" * 80)
print(classification_report(y_val, final_predictions))
print("-" * 80)

# Generate and plot confusion matrix
print("\n2. Generating Confusion Matrix...")
cm = confusion_matrix(y_val, final_predictions)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,
            xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])
plt.title('Confusion Matrix - Final Tuned RBF SVM', fontsize=14, fontweig
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()

print(f"   True Negatives:  {cm[0, 0]}")
print(f"   False Positives: {cm[0, 1]}")
print(f"   False Negatives: {cm[1, 0]}")
```

```
print(f"   True Positives:  {cm[1, 1]}")

# 2. Visualize Decision Boundaries
print("\n3. Visualizing Decision Boundaries...")

def plot_decision_boundary(model, X, y, title, ax):
    """Helper function to plot decision boundaries"""
    h = 0.02  # step size in the mesh
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # Predict for each point in the mesh
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
    ax.contourf(xx, yy, Z, alpha=0.3, cmap='RdYlBu')

    # Plot training points
    ax.scatter(X[y==0][:, 0], X[y==0][:, 1], c='blue',
               label='Class 0', alpha=0.6, edgecolors='k', s=50)
    ax.scatter(X[y==1][:, 0], X[y==1][:, 1], c='red',
               label='Class 1', alpha=0.6, edgecolors='k', s=50)

    ax.set_title(title, fontsize=12, fontweight='bold')
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.legend()
    ax.grid(True, alpha=0.3)

# Create 1x3 subplot
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Plot 1: Linear SVM
plot_decision_boundary(linear_model, X_train_scaled, y_train,
                       f'Linear SVM\n(Accuracy: {linear_accuracy:.4f})', a

# Plot 2: Default RBF SVM
plot_decision_boundary(rbf_model, X_train_scaled, y_train,
                       f'RBF SVM (Default)\n(Accuracy: {rbf_accuracy:.4f})

# Plot 3: Tuned RBF SVM (Best from GridSearch)
plot_decision_boundary(grid.best_estimator_, X_train_scaled, y_train,
                       f'Tuned RBF SVM (C={grid.best_params_["C"]}, γ={gri

plt.tight_layout()
plt.show()
```

```
1. Evaluating the best model on validation set...

   Final Validation Accuracy: 0.9667

   FINAL CLASSIFICATION REPORT:
--------------------------------------------------------------------------------
------
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        75
           1       1.00      0.93      0.97        75

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150

--------------------------------------------------------------------------------
------

2. Generating Confusion Matrix...
```




```
   True Negatives:  75
   False Positives: 0
   False Negatives: 5
   True Positives:  70

3. Visualizing Decision Boundaries...
```

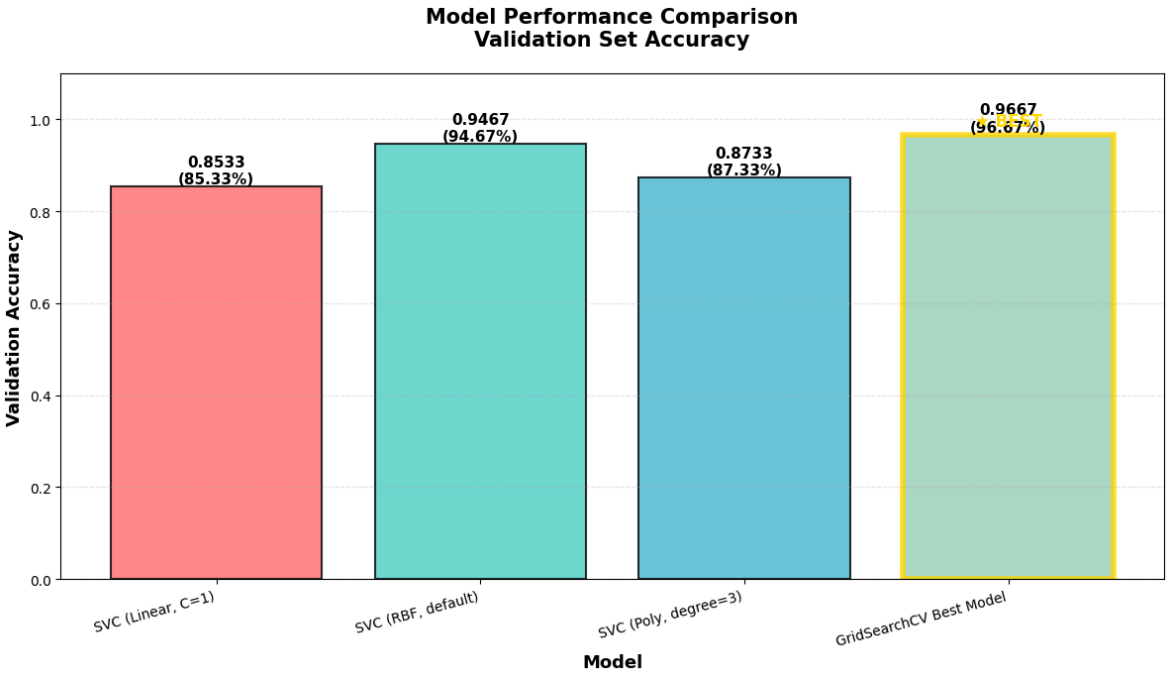


FINAL SUMMARY AND CONCLUSIONS

In [9]:
```
print("FINAL MODEL COMPARISON TABLE (FOR REPORT)")
print("=" * 80)

# Calculate final validation accuracy using the best model from grid sear
final_val_predictions = grid.predict(X_val_scaled)
final_val_accuracy = accuracy_score(y_val, final_val_predictions)

final_comparison = {
    'Model': [
        'SVC (Linear, C=1)',
        'SVC (RBF, default)',
        'SVC (Poly, degree=3)',
        'GridSearchCV Best Model'
    ],
    'Kernel': ['linear', 'rbf', 'poly', 'rbf'],
    'Key Parameters': [
        'C=1.0',
        'C=1.0, gamma=scale',
        'C=1.0, degree=3',
        f"C={grid.best_params_['C']}, gamma={grid.best_params_['gamma']}"
    ],
    'Validation Accuracy': [
        f"{linear_accuracy:.4f}",
        f"{rbf_accuracy:.4f}",
        f"{poly_accuracy:.4f}",
        f"{final_val_accuracy:.4f}"
    ],
    'Accuracy (%)': [
        f"{linear_accuracy*100:.2f}%",
        f"{rbf_accuracy*100:.2f}%",
        f"{poly_accuracy*100:.2f}%",
        f"{final_val_accuracy*100:.2f}%"
    ]
}
final_df = pd.DataFrame(final_comparison)
print(final_df.to_string(index=False))
print("=" * 80)

# Visual bar chart comparison
print("\n Model Performance Comparison Chart")
plt.figure(figsize=(12, 7))
models = final_comparison['Model']
accuracies = [linear_accuracy, rbf_accuracy, poly_accuracy, final_val_acc
colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4']

bars = plt.bar(range(len(models)), accuracies, color=colors, alpha=0.8, e
```

```
# Add value labels on bars
for i, (bar, acc) in enumerate(zip(bars, accuracies)):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height,
             f'{acc:.4f}\n({acc*100:.2f}%)',
             ha='center', va='bottom', fontsize=11, fontweight='bold')

# Highlight the best model
best_idx = accuracies.index(max(accuracies))
bars[best_idx].set_edgecolor('gold')
bars[best_idx].set_linewidth(4)
plt.text(best_idx, accuracies[best_idx] + 0.02, '★ BEST',
         ha='center', fontsize=12, fontweight='bold', color='gold')

plt.xlabel('Model', fontsize=13, fontweight='bold')
plt.ylabel('Validation Accuracy', fontsize=13, fontweight='bold')
plt.title('Model Performance Comparison\nValidation Set Accuracy',
          fontsize=15, fontweight='bold', pad=20)
plt.xticks(range(len(models)), models, rotation=15, ha='right')
plt.ylim(0, 1.1)
plt.grid(axis='y', alpha=0.3, linestyle='--')
plt.tight_layout()
plt.show()
```

```
FINAL MODEL COMPARISON TABLE (FOR REPORT)
================================================================================
======
                  Model Kernel     Key Parameters Validation Accuracy Accu
racy (%)
      SVC (Linear, C=1) linear              C=1.0              0.8533
85.33%
     SVC (RBF, default)    rbf C=1.0, gamma=scale              0.9467
94.67%
   SVC (Poly, degree=3)   poly    C=1.0, degree=3              0.8733
87.33%
GridSearchCV Best Model    rbf       C=1, gamma=1              0.9667
96.67%
================================================================================
======

 Model Performance Comparison Chart
```


Model Performance Comparison
Validation Set Accuracy
Validation Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0.8533
(85.33%)
0.9467
(94.67%)
0.8733
(87.33%)
0.9667
(96.67%)
SVC (Linear, C=1)
SVC (RBF, default)
SVC (Poly, degree=3)
GridSearchCV Best Model
Model