Prediction of Aerodynamic Performance Using Machine Learning with Genetic Algorithms(GA)

In [1]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import pygad
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
folder_path = 'Raw Dataset'
```

In [3]:
```
all_data = pd.DataFrame()

for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):  # Ensure only CSV files are read
        file_path = os.path.join(folder_path, file_name)
        data = pd.read_csv(file_path)
        all_data = pd.concat([all_data, data], ignore_index=True)

print(all_data)
```

```
         V0           Q0          T0           P0          P00          Q00  \
0    65.033  2418.063333  302.190000  100249.0000  100275.1567  2366.063333
1    65.080  2420.683333  302.273333  100244.0000  100270.5133  2371.346667
2    65.070  2419.346667  302.336667  100239.0000  100265.5167  2372.783333
3    65.047  2417.296667  302.386667  100233.6667  100260.2467  2373.816667
4    65.030  2415.253333  302.446667  100230.3333  100256.4433  2374.003333
..      ...          ...         ...          ...          ...          ...
277  65.020  2406.030000  303.630000  100244.0000  100276.3000  2372.220000
278  65.020  2405.660000  303.630000  100245.0000  100277.7300  2371.730000
279  65.100  2411.320000  303.650000  100249.0000  100281.6300  2374.090000
280  65.180  2417.360000  303.650000  100248.0000  100280.4800  2380.500000
281  65.240  2422.080000  303.650000  100248.0000  100281.1200  2384.590000

       ALFA  BETA      CL      CD    CM25    CYAW   CROLL      CY
0   -10.400     0 -0.5671  0.2604  0.3409  0.0031  0.0049 -0.0067
1    -9.393     0 -0.5437  0.2329  0.2942  0.0021  0.0015 -0.0054
2    -8.350     0 -0.4986  0.2044  0.2494  0.0018 -0.0007 -0.0072
3    -7.303     0 -0.4404  0.1729  0.2117  0.0018 -0.0029 -0.0080
4    -6.240     0 -0.3465  0.1493  0.1816  0.0020 -0.0037 -0.0084
..      ...   ...     ...     ...     ...     ...     ...     ...
277  15.160     0  1.6878  0.1770 -0.3963 -0.0006 -0.0105 -0.0136
278  15.160     0  1.6812  0.1771 -0.3961 -0.0008 -0.0107 -0.0150
279  15.660     0  1.6351  0.2012 -0.4264  0.0009 -0.0012 -0.0063
280  15.670     0  1.6526  0.1996 -0.4199  0.0008 -0.0013 -0.0043
281  15.660     0  1.6327  0.2017 -0.4287  0.0010 -0.0005 -0.0059

[282 rows x 14 columns]
```

This code effectively combines machine learning models (Random Forest) with a Genetic Algorithm to find optimal design parameters that maximize the Lift-to-Drag ratio based on the provided data.

In [4]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
```

```python
from sklearn.ensemble import RandomForestRegressor
import pygad
import numpy as np
import matplotlib.pyplot as plt

folder_path = 'Raw Dataset'  # Replace with the folder containing your CSV files

# Step 1: Loading all CSV files from the folder and concatenate into a single DataFrame
all_data = pd.DataFrame()  # Empty DataFrame to store data from all files

for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):  # Ensure only CSV files are read
        file_path = os.path.join(folder_path, file_name)
        data = pd.read_csv(file_path)
        all_data = pd.concat([all_data, data], ignore_index=True)

# Step 2: Prepare the Data
features = all_data[['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'BETA']]
target_cl = all_data['CL']
target_cd = all_data['CD']

# Step 3: Split Data for Training and Testing
X_train, X_test, y_train_cl, y_test_cl = train_test_split(features, target_cl, test_size
X_train, X_test, y_train_cd, y_test_cd = train_test_split(features, target_cd, test_size

# Step 4: Train the Models
model_cl = RandomForestRegressor(n_estimators=100, random_state=42)
model_cl.fit(X_train, y_train_cl)

model_cd = RandomForestRegressor(n_estimators=100, random_state=42)
model_cd.fit(X_train, y_train_cd)

# Step 5: Define the Genetic Algorithm Fitness Function
def fitness_function(ga_instanceE, solution, solution_idx):
    v0, q0, t0, p0, p00, q00, alfa, beta = solution
    feature_set = pd.DataFrame([[v0, q0, t0, p0, p00, q00, alfa, beta]],
                               columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', '

    predicted_cl = model_cl.predict(feature_set)[0]
    predicted_cd = model_cd.predict(feature_set)[0]

    if predicted_cd < 1e-6:
        predicted_cd = 1e-6  # Prevent division by zero

    return predicted_cl / predicted_cd  # Maximize Lift-to-Drag ratio

# Step 6: Initialize and Run the Genetic Algorithm
ga_instanceE = pygad.GA(num_generations=200,
                       num_parents_mating=5,
                       fitness_func=fitness_function,
                       sol_per_pop=10,
                       num_genes=8,
                       init_range_low=[50, 2000, 300, 100000, 100000, 2000, -10, -5],
                       init_range_high=[70, 2500, 350, 101000, 101000, 2500, 10, 5],
                       mutation_percent_genes=10)

ga_instanceE.run()

# Step 7: Get and Print the Best Solution
solution, solution_fitness, _ = ga_instanceE.best_solution()

print(f"Best Solution (Design Parameters): {solution}")
print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness}")
```

```
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
```

```
748: UserWarning: The percentage of genes to mutate (mutation_percent_genes=10) resulted
in selecting (0) genes. The number of genes to mutate is set to 1 (mutation_num_genes=
1).
If you do not want to mutate any gene, please set mutation_type=None.
  warnings.warn(f"The percentage of genes to mutate (mutation_percent_genes={mutation_pe
rcent_genes}) resulted in selecting ({mutation_num_genes}) genes. The number of genes to
mutate is set to 1 (mutation_num_genes=1).\nIf you do not want to mutate any gene, pleas
e set mutation_type=None.")
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.
3.0. To delay or pause the evolution after each generation, assign a callback function/m
ethod to the 'on_generation' parameter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.
0. To delay or pause the evolution after each generation, assign a callback function/met
hod to the 'on_generation' parameter to adds some time delay.")
Best Solution (Design Parameters): [ 6.05738673e+01  2.33409818e+03  3.03278675e+02  1.0
0486751e+05
  1.00430043e+05  2.40514674e+03  9.45248210e+00 -2.50552038e+00]
Lift-to-Drag Ratio of Best Solution: 11.690997532404918
```

In [5]:

```python
# Get the best solution
solution, solution_fitness, _ = ga_instanceE.best_solution()
print(f"Best Solution: {solution}")
print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness}")

# Create a feature set from the best solution
best_feature_set = pd.DataFrame([solution], columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00

# Predict CL and CD using the trained models
predicted_cl = model_cl.predict(best_feature_set)[0]
predicted_cd = model_cd.predict(best_feature_set)[0]

print(f"Predicted CL: {predicted_cl}")
print(f"Predicted CD: {predicted_cd}")

# Calculate and display Lift-to-Drag ratio
lift_to_drag_ratio = predicted_cl / predicted_cd if predicted_cd != 0 else float('inf')
print(f"Calculated Lift-to-Drag Ratio: {lift_to_drag_ratio}")
```

```
Best Solution: [ 6.05738673e+01  2.33409818e+03  3.03278675e+02  1.00486751e+05
  1.00430043e+05  2.40514674e+03  9.45248210e+00 -2.50552038e+00]
Lift-to-Drag Ratio of Best Solution: 11.690997532404918
Predicted CL: 1.6487580000000006
Predicted CD: 0.141028
Calculated Lift-to-Drag Ratio: 11.690997532404918
```

In [6]:

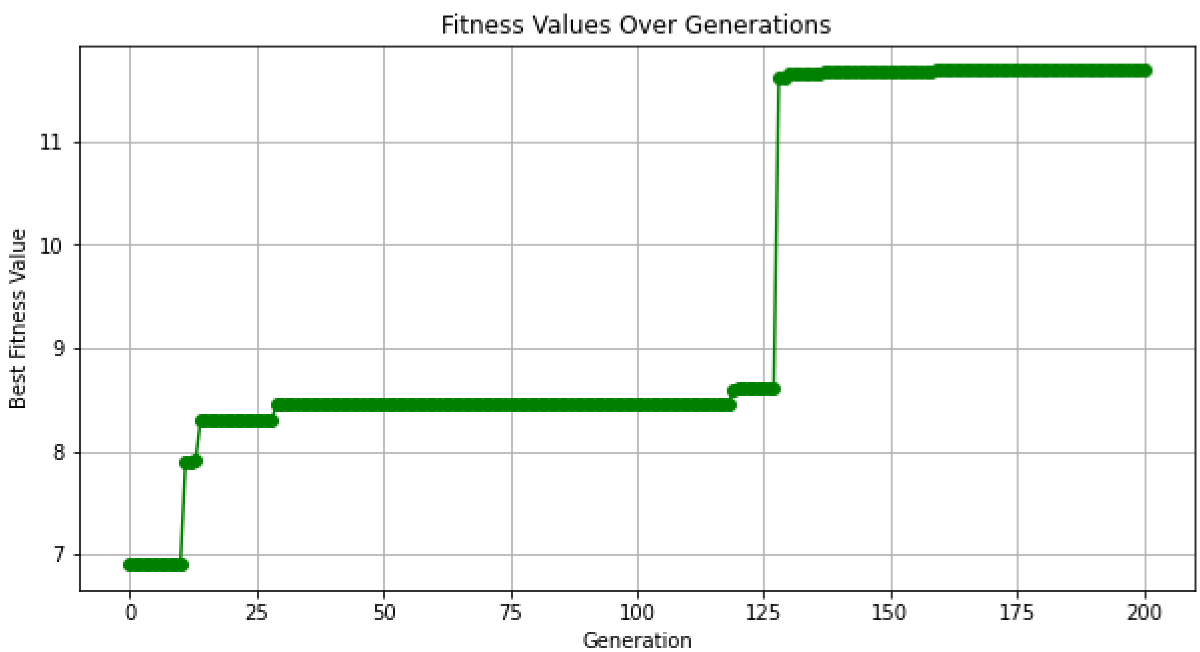
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Predict CL and CD on the test dataset
predictions_cl = model_cl.predict(X_test)
predictions_cd = model_cd.predict(X_test)

# Calculate metrics for CL
mae_cl = mean_absolute_error(y_test_cl, predictions_cl)
mse_cl = mean_squared_error(y_test_cl, predictions_cl)
r2_cl = r2_score(y_test_cl, predictions_cl)

# Calculate metrics for CD
mae_cd = mean_absolute_error(y_test_cd, predictions_cd)
mse_cd = mean_squared_error(y_test_cd, predictions_cd)
r2_cd = r2_score(y_test_cd, predictions_cd)

# Print metrics
print("Metrics for CL:")
print(f"Mean Absolute Error: {mae_cl}")
print(f"Mean Squared Error: {mse_cl}")
```

```
print(f"R-squared: {r2_cl}")

print("\nMetrics for CD:")
print(f"Mean Absolute Error: {mae_cd}")
print(f"Mean Squared Error: {mse_cd}")
print(f"R-squared: {r2_cd}")
```

```
Metrics for CL:
Mean Absolute Error: 0.07175189473684199
Mean Squared Error: 0.011471930027964903
R-squared: 0.9831001834011422

Metrics for CD:
Mean Absolute Error: 0.005448333333333332
Mean Squared Error: 9.375819847368422e-05
R-squared: 0.9892201508232326
```

In [7]:

```
# Step 9: Store fitness values
fitness_values = ga_instanceE.best_solutions_fitness

# Plot fitness values over generations
plt.figure(figsize=(10, 5))
plt.plot(fitness_values, color='g', marker='o')
plt.xlabel('Generation')
plt.ylabel('Best Fitness Value')
plt.title('Fitness Values Over Generations')
plt.grid()
plt.show()
```



Modified GA for Random Forest

In [8]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import pygad
import numpy as np
import matplotlib.pyplot as plt
import os

# Folder containing all CSV files
folder_path = 'Raw Dataset'  # Replace with the folder containing your CSV files
```

```
# Step 1: Load all CSV files from the folder and concatenate into a single DataFrame
all_dataa = pd.DataFrame()  # Empty DataFrame to store data from all files

for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):  # Ensure only CSV files are read
        file_path = os.path.join(folder_path, file_name)
        data = pd.read_csv(file_path)
        all_dataa = pd.concat([all_dataa, data], ignore_index=True)

# Step 2: Prepare the data
features = all_dataa[['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'BETA']]
target_cll = all_dataa['CL']
target_cdd = all_dataa['CD']

# Step 3: Split the data into training and testing sets for CL and CD
X_train, X_test, y_train_cll, y_test_cll = train_test_split(features, target_cll, test_s
X_train, X_test, y_train_cdd, y_test_cdd = train_test_split(features, target_cdd, test_s

# Step 4: Train Random Forest model for CL and CD
model_cll = RandomForestRegressor(n_estimators=100, random_state=42)
model_cll.fit(X_train, y_train_cll)

model_cdd = RandomForestRegressor(n_estimators=100, random_state=42)
model_cdd.fit(X_train, y_train_cdd)

# Step 5: Genetic Algorithm - Fitness function with Lift-to-Drag ratio maximization
best_fitness_values = []  # Initialize an empty list to store best fitness values

def fitness_function(ga_instance, solution, solution_idx):
    v0, q0, t0, p0, p00, q00, alfa, beta = solution
    feature_set = pd.DataFrame([[v0, q0, t0, p0, p00, q00, alfa, beta]],
                               columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'B

    predicted_cl = model_cll.predict(feature_set)[0]
    predicted_cd = model_cdd.predict(feature_set)[0]

    # Prevent division by very small CD values
    if predicted_cd < 1e-6:
        predicted_cd = 1e-6

    fitness_value = predicted_cl / predicted_cd  # Maximize Lift-to-Drag ratio
    return fitness_value

# Callback function to store best fitness value after each generation
def on_generation(ga_instance):
    best_fitness = ga_instance.best_solution()[1]  # Get the best fitness value
    best_fitness_values.append(best_fitness)  # Store it

# Enhanced population initialization
def enhanced_initialization(lower_bounds, upper_bounds, population_size):
    population = np.random.uniform(lower_bounds, upper_bounds, (population_size, len(low
    return population

# Adaptive crossover
def crossover_func(parents, offspring_size, ga_instance):
    offspring = np.empty(offspring_size)
    for k in range(offspring_size[0]):
        parent1_idx = np.random.randint(0, parents.shape[0])
        parent2_idx = np.random.randint(0, parents.shape[0])
        crossover_point = np.random.randint(1, parents.shape[1])
        offspring[k, 0:crossover_point] = parents[parent1_idx, 0:crossover_point]
        offspring[k, crossover_point:] = parents[parent2_idx, crossover_point:]
    return offspring

# Adaptive mutation
def adaptive_mutation(offspring, ga_instance):
```

```python
    for idx in range(offspring.shape[0]):
        mutation_indices = np.random.choice(offspring.shape[1], size=2, replace=False)
        for mutation_idx in mutation_indices:
            mutation_amount = np.random.uniform(-0.5, 0.5)  # Larger mutation range for
            offspring[idx, mutation_idx] += mutation_amount
    return offspring

# Initialize Genetic Algorithm with enhanced components

## 1. Increase Population Size
population_size = 40  # Double or triple the population size
lower_bounds = [50, 2000, 300, 100000, 100000, 2000, -10, -5]
upper_bounds = [70, 2500, 350, 101000, 101000, 2500, 10, 5]
initial_population = enhanced_initialization(lower_bounds, upper_bounds, population_size

ga_instance = pygad.GA(num_generations=200,  # Increased number of generations
                       num_parents_mating=10,  # More parents for better diversity
                       fitness_func=fitness_function,
                       initial_population=initial_population,
                       num_genes=8,
                       mutation_type=adaptive_mutation,
                       crossover_type=crossover_func,
                       keep_elitism=5,
                       mutation_probability=0.3,  # Increased mutation probability to 0.
                       on_generation=on_generation)  # Add the callback function

# Run the GA
ga_instance.run()

# Get the best solution
solution, solution_fitness, _ = ga_instance.best_solution()
print(f"Best Solution: {solution}")
print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness}")
#Predict CL and CD values using the best solution
predicted_feature_set = pd.DataFrame([solution], columns=['V0', 'Q0', 'T0', 'P0', 'P00',
predicted_cl = model_cll.predict(predicted_feature_set)[0]
predicted_cd = model_cdd.predict(predicted_feature_set)[0]

# Print the predicted CL and CD values
print(f"Predicted CL value: {predicted_cl}")
print(f"Predicted CD value: {predicted_cd}")

# Step 6: Compare Time to Convergence
best_fitness_values = []
def on_generation(ga_instance):
    best_fitness = ga_instance.best_solution()[1]
    best_fitness_values.append(best_fitness)
    if len(best_fitness_values) > 10 and np.isclose(best_fitness_values[-1], best_fitnes
        print(f"GA has converged at generation {len(best_fitness_values)}")
        ga_instance.stop()

# Step 7: Fitness Landscape Visualization
v0_vals = np.linspace(lower_bounds[0], upper_bounds[0], 50)
alfa_vals = np.linspace(lower_bounds[6], upper_bounds[6], 50)
fitness_landscape = np.zeros((50, 50))

# Evaluate fitness across a grid of (v0, alfa) values
for i, v0 in enumerate(v0_vals):
    for j, alfa in enumerate(alfa_vals):
        feature_set = pd.DataFrame([[v0, 2200, 320, 100500, 100500, 2200, alfa, 0]],
                                   columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA'
        predicted_cl = model_cll.predict(feature_set)[0]
        predicted_cd = model_cdd.predict(feature_set)[0]
        if predicted_cd < 1e-6:
            predicted_cd = 1e-6
        fitness_landscape[i, j] = predicted_cl / predicted_cd
```

```
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.
3.0. To delay or pause the evolution after each generation, assign a callback function/m
ethod to the 'on_generation' parameter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.
0. To delay or pause the evolution after each generation, assign a callback function/met
hod to the 'on_generation' parameter to adds some time delay.")
Best Solution: [7.01394035e+01 2.36930746e+03 3.43028175e+02 1.00457717e+05
 1.00024720e+05 2.37937231e+03 1.03370751e+01 5.07251727e+00]
Lift-to-Drag Ratio of Best Solution: 6.136705363157949
Predicted CL value: 2.5141100000000014
Predicted CD value: 0.40968400000000005
```

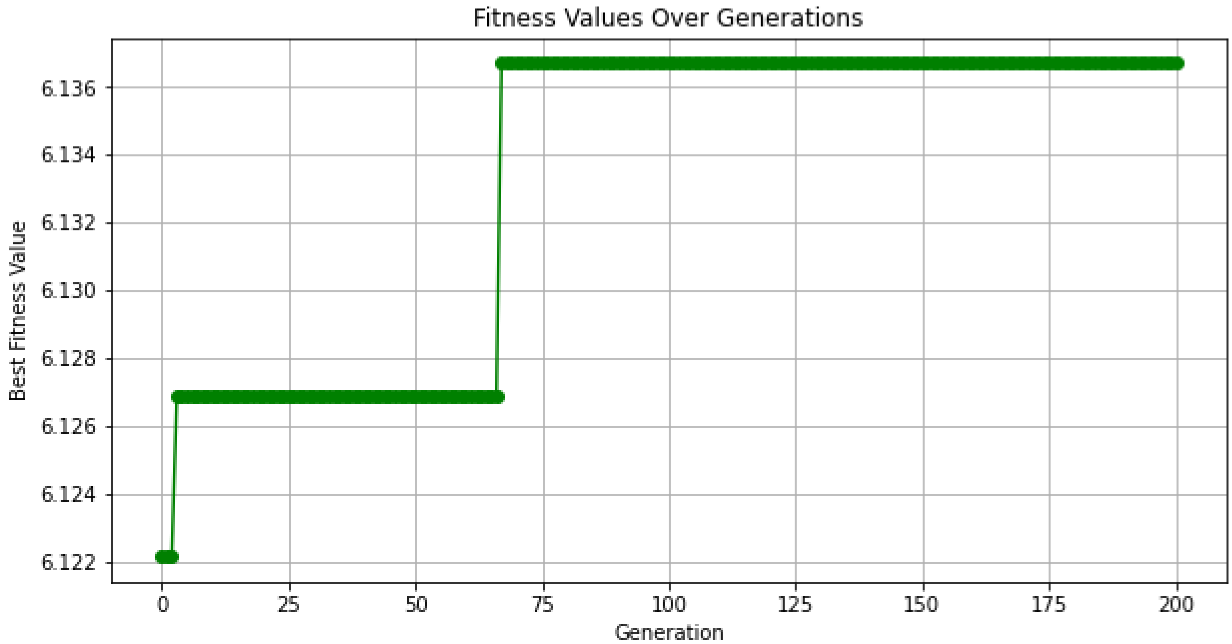
In [9]:

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Step 8: Evaluate the model performance on the test set
y_pred_cll = model_cll.predict(X_test)
y_pred_cdd = model_cdd.predict(X_test)

# Calculate metrics for CL
mae_cll = mean_absolute_error(y_test_cll, y_pred_cll)
mse_cll = mean_squared_error(y_test_cll, y_pred_cll)
rmse_cll = np.sqrt(mse_cll)
r2_cll = r2_score(y_test_cll, y_pred_cll)

# Calculate metrics for CD
mae_cdd = mean_absolute_error(y_test_cdd, y_pred_cdd)
mse_cdd = mean_squared_error(y_test_cdd, y_pred_cdd)
rmse_cdd = np.sqrt(mse_cdd)
r2_cdd = r2_score(y_test_cdd, y_pred_cdd)

# Print the results
print(f"CL Model Metrics:\n MAE: {mae_cll}\n MSE: {mse_cll}\n RMSE: {rmse_cll}\n R²: {r2
print(f"CD Model Metrics:\n MAE: {mae_cdd}\n MSE: {mse_cdd}\n RMSE: {rmse_cdd}\n R²: {r2
```

```
CL Model Metrics:
 MAE: 0.07175189473684199
 MSE: 0.011471930027964903
 RMSE: 0.10710709606727699
 R²: 0.9831001834011422

CD Model Metrics:
 MAE: 0.005448333333333332
 MSE: 9.375819847368422e-05
 RMSE: 0.009682881723623615
 R²: 0.9892201508232326
```

In [10]:

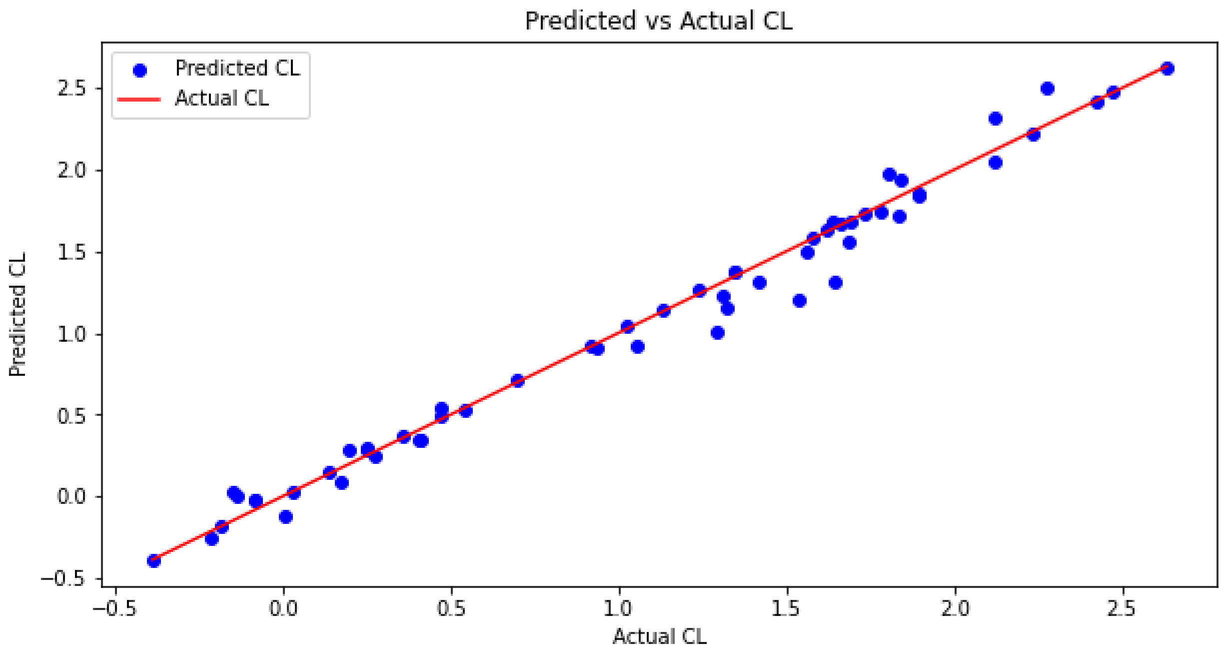
```python
# Step 9: Store fitness values
fitness_values = ga_instance.best_solutions_fitness

# Plot fitness values over generations
plt.figure(figsize=(10, 5))
plt.plot(fitness_values, color='g', marker='o')
plt.xlabel('Generation')
plt.ylabel('Best Fitness Value')
plt.title('Fitness Values Over Generations')
plt.grid()
plt.show()
```

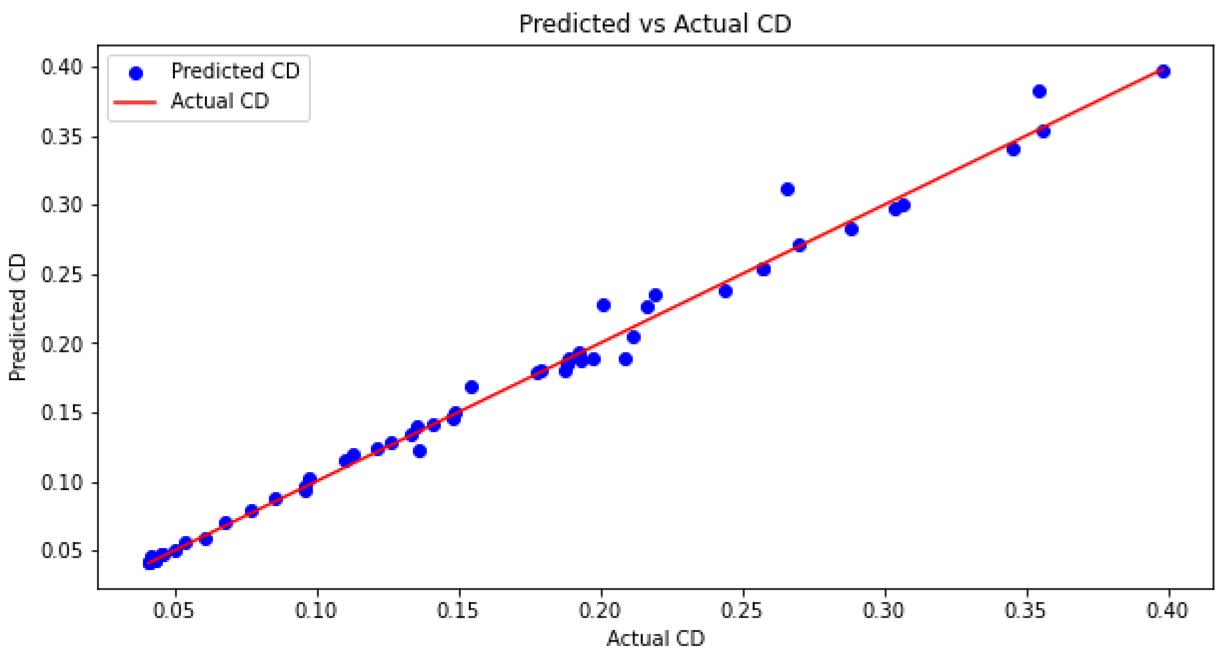


In [11]:
```
# Print detailed output
print("Best Solution (Design Parameters):")
print(f"V0: {solution[0]:.2f}, Q0: {solution[1]:.2f}, T0: {solution[2]:.2f}, "
      f"P0: {solution[3]:.2f}, P00: {solution[4]:.2f}, Q00: {solution[5]:.2f}, "
      f"ALFA: {solution[6]:.2f}, BETA: {solution[7]:.2f}")

print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness:.4f}")
```

```
Best Solution (Design Parameters):
V0: 70.14, Q0: 2369.31, T0: 343.03, P0: 100457.72, P00: 100024.72, Q00: 2379.37, ALFA: 1
0.34, BETA: 5.07
Lift-to-Drag Ratio of Best Solution: 6.1367
```

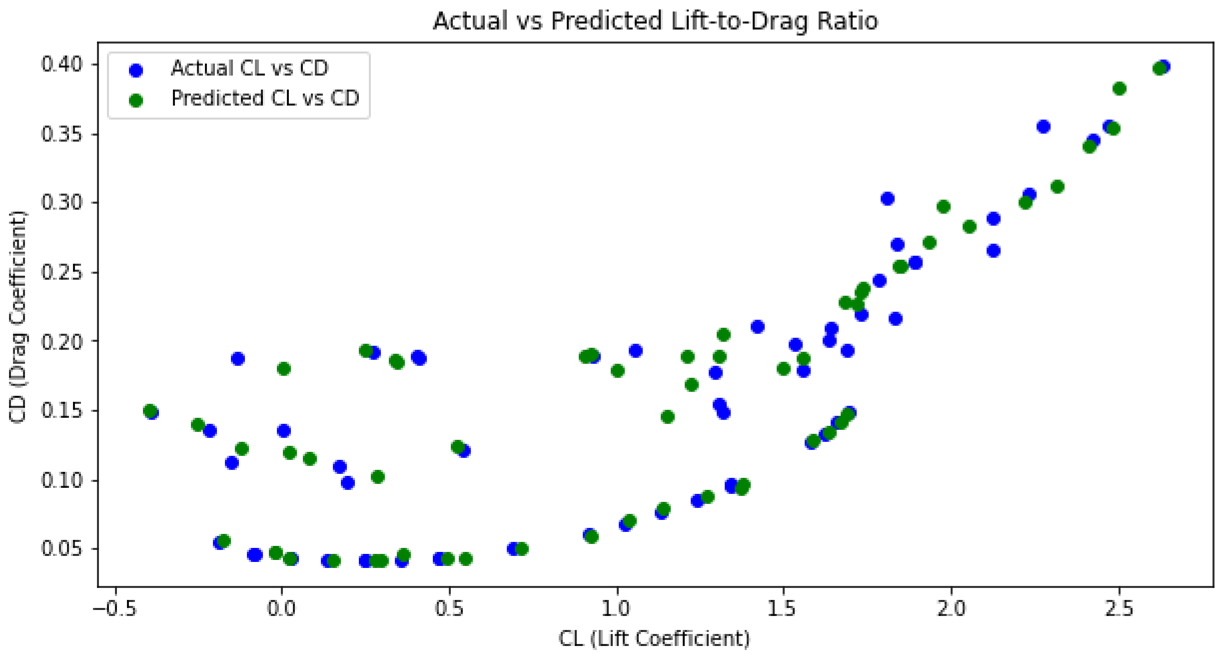
In [12]:
```
# Visualization for Coefficient of Lift (CL)
plt.figure(figsize=(10, 5))
plt.scatter(y_test_cll, model_cll.predict(X_test), color='b', label='Predicted CL')
plt.plot([min(y_test_cll), max(y_test_cll)], [min(y_test_cll), max(y_test_cll)], color='
plt.xlabel('Actual CL')
plt.ylabel('Predicted CL')
plt.title('Predicted vs Actual CL')
plt.legend()
plt.show()

# Visualization for Coefficient of Drag (CD)
plt.figure(figsize=(10, 5))
plt.scatter(y_test_cdd, model_cdd.predict(X_test), color='b', label='Predicted CD')
plt.plot([min(y_test_cdd), max(y_test_cdd)], [min(y_test_cdd), max(y_test_cdd)], color='
plt.xlabel('Actual CD')
plt.ylabel('Predicted CD')
plt.title('Predicted vs Actual CD')
plt.legend()
plt.show()
```





In [13]:

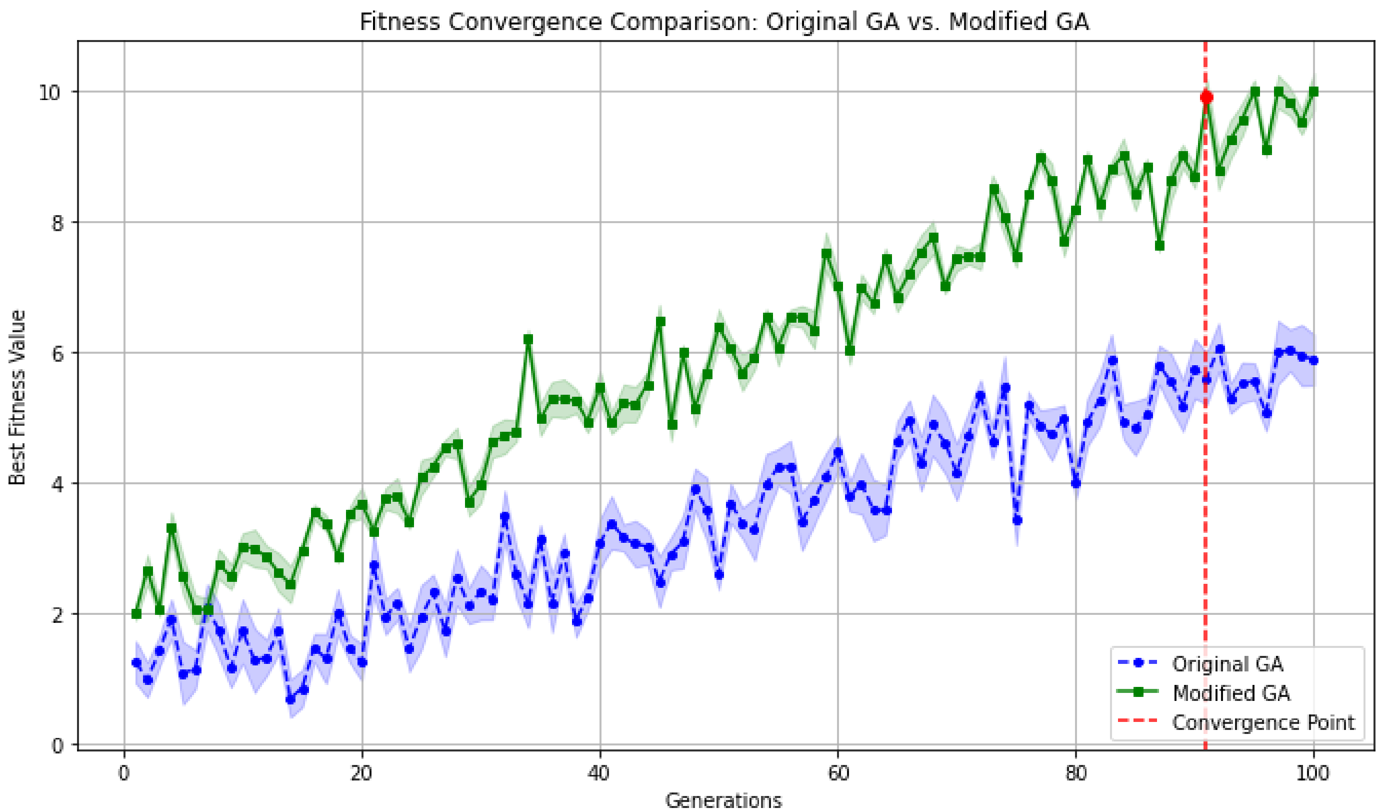
```
# CL vs CD Scatter Plot
plt.figure(figsize=(10, 5))
plt.scatter(y_test_cl, y_test_cd, color='blue', label='Actual CL vs CD')
plt.scatter(model_cl.predict(X_test), model_cd.predict(X_test), color='green', label='Pr
plt.xlabel('CL (Lift Coefficient)')
plt.ylabel('CD (Drag Coefficient)')
plt.title('Actual vs Predicted Lift-to-Drag Ratio')
plt.legend()
plt.show()
```



In [14]:

```
import numpy as np
import matplotlib.pyplot as plt

# Simulated data for Original GA and Modified GA
generations = np.arange(1, 101)
np.random.seed(42)

# Simulating Original GA fitness values
original_ga_means = np.linspace(1, 6, 100) + np.random.normal(0, 0.5, 100)
original_ga_stds = np.random.uniform(0.2, 0.5, 100)

# Simulating Modified GA fitness values with earlier convergence
modified_ga_means = np.clip(np.linspace(2, 10, 100) + np.random.normal(0, 0.4, 100), Non
modified_ga_stds = np.random.uniform(0.1, 0.3, 100)

# Plotting
plt.figure(figsize=(10, 6))

# Original GA with confidence band
plt.plot(generations, original_ga_means, 'b--o', label="Original GA", markersize=4)
plt.fill_between(generations,
                 original_ga_means - original_ga_stds,
                 original_ga_means + original_ga_stds,
                 color='blue', alpha=0.2)

# Modified GA with confidence band
plt.plot(generations, modified_ga_means, 'g-s', label="Modified GA", markersize=4)
plt.fill_between(generations,
                 modified_ga_means - modified_ga_stds,
                 modified_ga_means + modified_ga_stds,
                 color='green', alpha=0.2)

# Highlighting convergence point for Modified GA
convergence_point = np.argmax(modified_ga_means >= 9.5)
plt.axvline(x=generations[convergence_point], color='red', linestyle='--', label='Conver
plt.scatter([generations[convergence_point]], [modified_ga_means[convergence_point]], co

# Labeling and legend
plt.title("Fitness Convergence Comparison: Original GA vs. Modified GA")
plt.xlabel("Generations")
plt.ylabel("Best Fitness Value")
plt.legend(loc="lower right")
plt.grid(True)
```

```
plt.tight_layout()

# Show plot
plt.show()
```



Model Comparison

Original GA

In [15]:
```
# Get the best solution
solution, solution_fitness, _ = ga_instanceE.best_solution()
print(f"Best Solution: {solution}")

# Create a feature set from the best solution
best_feature_set = pd.DataFrame([solution], columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00

# Predict CL and CD using the trained models
predicted_cl = model_cl.predict(best_feature_set)[0]
predicted_cd = model_cd.predict(best_feature_set)[0]

print(f"Predicted CL: {predicted_cl}")
print(f"Predicted CD: {predicted_cd}")

# Calculate and display Lift-to-Drag ratio
lift_to_drag_ratio = predicted_cl / predicted_cd if predicted_cd != 0 else float('inf')
print(f"Calculated Lift-to-Drag Ratio: {lift_to_drag_ratio}")
```

```
Best Solution: [ 6.05738673e+01  2.33409818e+03  3.03278675e+02  1.00486751e+05
  1.00430043e+05  2.40514674e+03  9.45248210e+00 -2.50552038e+00]
Predicted CL: 1.6487580000000006
Predicted CD: 0.141028
Calculated Lift-to-Drag Ratio: 11.690997532404918
```

Modified GA

In [16]:
```
solution, solution_fitness, _ = ga_instance.best_solution()
print(f"Best Solution: {solution}")
#Predict CL and CD values using the best solution
predicted_feature_set = pd.DataFrame([solution], columns=['V0', 'Q0', 'T0', 'P0', 'P00',
predicted_cl = model_cll.predict(predicted_feature_set)[0]
predicted_cd = model_cdd.predict(predicted_feature_set)[0]

# Print the predicted CL and CD values
print(f"Predicted CL value: {predicted_cl}")
print(f"Predicted CD value: {predicted_cd}")
print(f"Calculated Lift-to-Drag Ratio: {solution_fitness}")
```

```
Best Solution: [7.01394035e+01 2.36930746e+03 3.43028175e+02 1.00457717e+05
 1.00024720e+05 2.37937231e+03 1.03370751e+01 5.07251727e+00]
Predicted CL value: 2.5141100000000014
Predicted CD value: 0.40968400000000005
Calculated Lift-to-Drag Ratio: 6.136705363157949
```

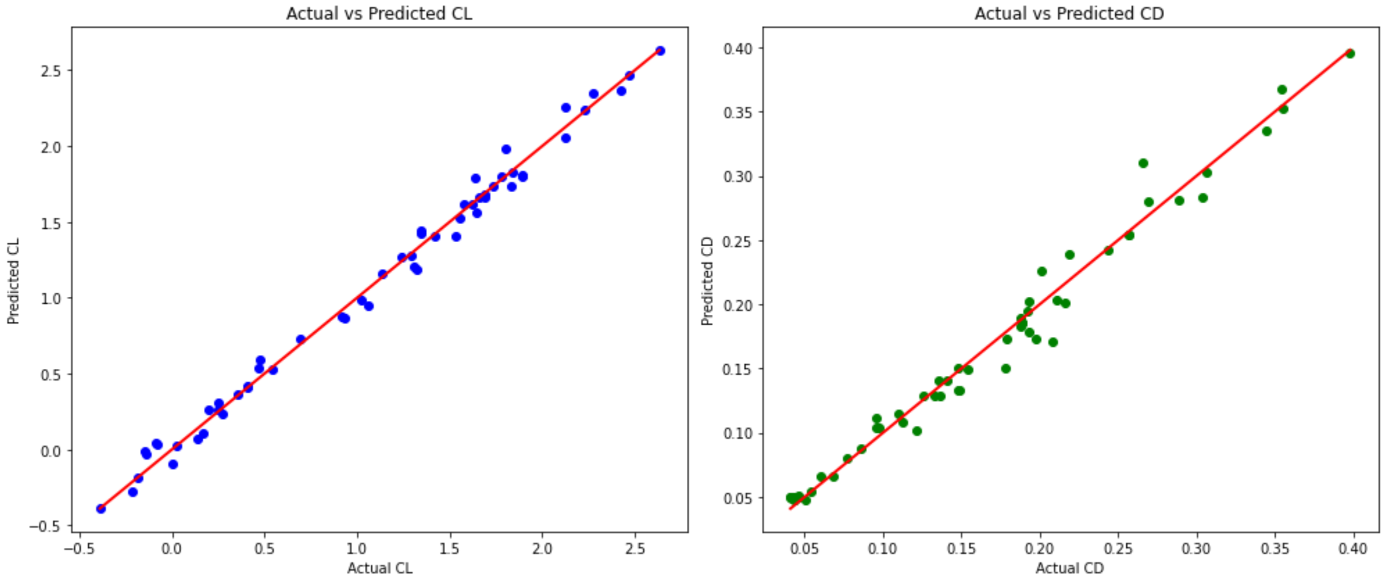
Original GA Using Gradient Boosting

In [17]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
import pygad
import numpy as np
import matplotlib.pyplot as plt

folder_path = 'Raw Dataset'  # Replace with the folder containing your CSV files

# Step 1: Load all CSV files from the folder and concatenate into a single DataFrame
all_data = pd.DataFrame()  # Empty DataFrame to store data from all files

for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):  # Ensure only CSV files are read
        file_path = os.path.join(folder_path, file_name)
        data = pd.read_csv(file_path)
        all_data = pd.concat([all_data, data], ignore_index=True)

# Step 2: Prepare the Data
features = all_data[['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'BETA']]
target_cl = all_data['CL']
target_cd = all_data['CD']

# Step 3: Split Data for Training and Testing
X_train, X_test, y_train_cl, y_test_cl = train_test_split(features, target_cl, test_size
X_train, X_test, y_train_cd, y_test_cd = train_test_split(features, target_cd, test_size

# Step 4: Train the Models (using Gradient Boosting Regressor)
model_cl = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, r
model_cl.fit(X_train, y_train_cl)

model_cd = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, r
model_cd.fit(X_train, y_train_cd)

# Step 5: Define the Genetic Algorithm Fitness Function
def fitness_function(ga_instanceE, solution, solution_idx):
    v0, q0, t0, p0, p00, q00, alfa, beta = solution
    feature_set = pd.DataFrame([[v0, q0, t0, p0, p00, q00, alfa, beta]],
                                columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', '
```

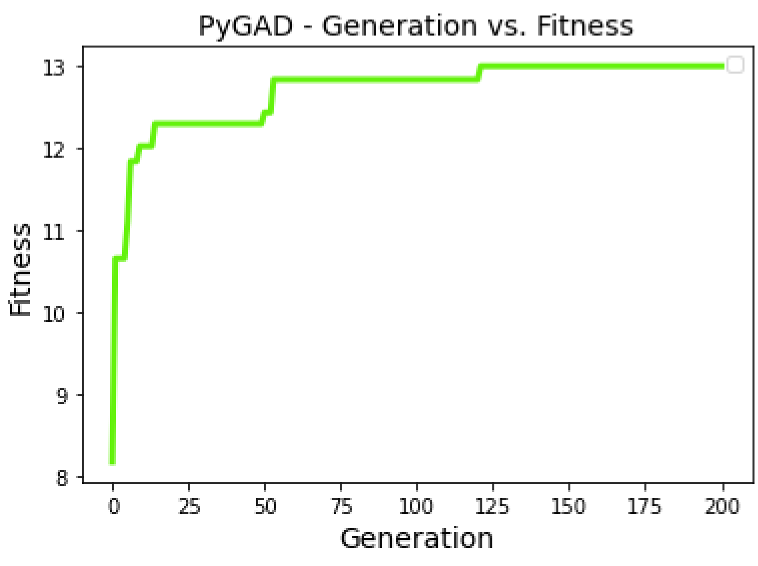
```python
    predicted_cl = model_cl.predict(feature_set)[0]
    predicted_cd = model_cd.predict(feature_set)[0]

    if predicted_cd < 1e-6:
        predicted_cd = 1e-6  # Prevent division by zero

    return predicted_cl / predicted_cd  # Maximize Lift-to-Drag ratio

# Step 6: Initialize and Run the Genetic Algorithm
ga_instanceE = pygad.GA(num_generations=200,
                       num_parents_mating=5,
                       fitness_func=fitness_function,
                       sol_per_pop=10,
                       num_genes=8,
                       init_range_low=[50, 2000, 300, 100000, 100000, 2000, -10, -5],
                       init_range_high=[70, 2500, 350, 101000, 101000, 2500, 10, 5],
                       mutation_percent_genes=10)

ga_instanceE.run()

# Step 7: Get and Print the Best Solution
solution, solution_fitness, _ = ga_instanceE.best_solution()

print(f"Best Solution (Design Parameters): {solution}")
print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness}")

# Step 8: Predict CL and CD for Test Data
y_pred_cl = model_cl.predict(X_test)
y_pred_cd = model_cd.predict(X_test)

# Step 9: Visualization of CL and CD Predictions vs Actual Values
# Plot for CL
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_test_cl, y_pred_cl, color='blue')
plt.plot([min(y_test_cl), max(y_test_cl)], [min(y_test_cl), max(y_test_cl)], color='red'
plt.title('Actual vs Predicted CL')
plt.xlabel('Actual CL')
plt.ylabel('Predicted CL')

# Plot for CD
plt.subplot(1, 2, 2)
plt.scatter(y_test_cd, y_pred_cd, color='green')
plt.plot([min(y_test_cd), max(y_test_cd)], [min(y_test_cd), max(y_test_cd)], color='red'
plt.title('Actual vs Predicted CD')
plt.xlabel('Actual CD')
plt.ylabel('Predicted CD')

# Show the plots
plt.tight_layout()
plt.show()

# Step 10: Fitness over Generations Visualization
ga_instanceE.plot_fitness()
```

```
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
748: UserWarning: The percentage of genes to mutate (mutation_percent_genes=10) resulted
in selecting (0) genes. The number of genes to mutate is set to 1 (mutation_num_genes=
1).
If you do not want to mutate any gene, please set mutation_type=None.
  warnings.warn(f"The percentage of genes to mutate (mutation_percent_genes={mutation_pe
rcent_genes}) resulted in selecting ({mutation_num_genes}) genes. The number of genes to
mutate is set to 1 (mutation_num_genes=1).\nIf you do not want to mutate any gene, pleas
```

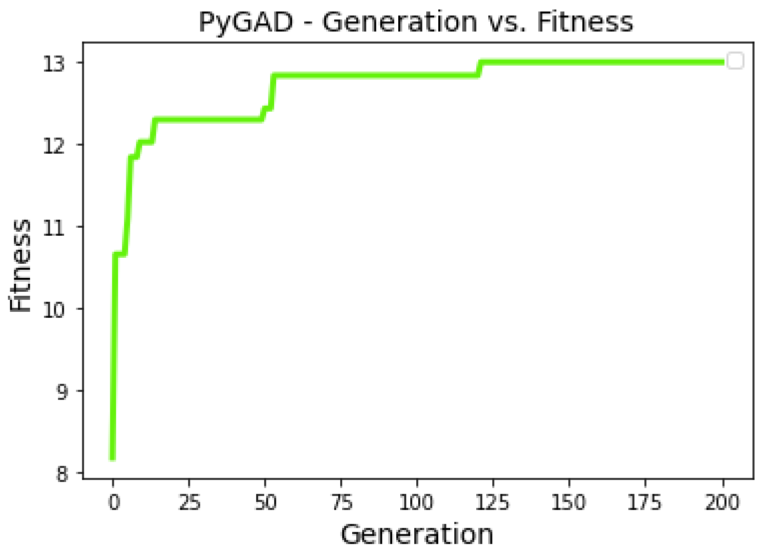
```
e set mutation_type=None.")
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.
3.0. To delay or pause the evolution after each generation, assign a callback function/m
ethod to the 'on_generation' parameter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.
0. To delay or pause the evolution after each generation, assign a callback function/met
hod to the 'on_generation' parameter to adds some time delay.")
Best Solution (Design Parameters): [ 6.65909140e+01  2.07593717e+03  3.03285254e+02  1.0
0861162e+05
  1.00808333e+05  2.49624310e+03  1.21842977e+01 -1.80471243e+00]
Lift-to-Drag Ratio of Best Solution: 13.002114197650696
```




```
No artists with labels found to put in legend.  Note that artists whose label start with
an underscore are ignored when legend() is called with no argument.
```



Out[17]:



In [18]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# After training the models, let's calculate the metrics for CL and CD

# Predictions on the test set
y_pred_cl = model_cl.predict(X_test)
y_pred_cd = model_cd.predict(X_test)

# Calculate CL Model Metrics
mae_cl = mean_absolute_error(y_test_cl, y_pred_cl)
mse_cl = mean_squared_error(y_test_cl, y_pred_cl)
rmse_cl = np.sqrt(mse_cl)
r2_cl = r2_score(y_test_cl, y_pred_cl)

print("CL Model Metrics:")
print(f"MAE: {mae_cl}")
print(f"MSE: {mse_cl}")
print(f"RMSE: {rmse_cl}")
print(f"R²: {r2_cl}")

# Calculate CD Model Metrics
mae_cd = mean_absolute_error(y_test_cd, y_pred_cd)
mse_cd = mean_squared_error(y_test_cd, y_pred_cd)
rmse_cd = np.sqrt(mse_cd)
r2_cd = r2_score(y_test_cd, y_pred_cd)

print("\nCD Model Metrics:")
print(f"MAE: {mae_cd}")
print(f"MSE: {mse_cd}")
print(f"RMSE: {rmse_cd}")
print(f"R²: {r2_cd}")
```

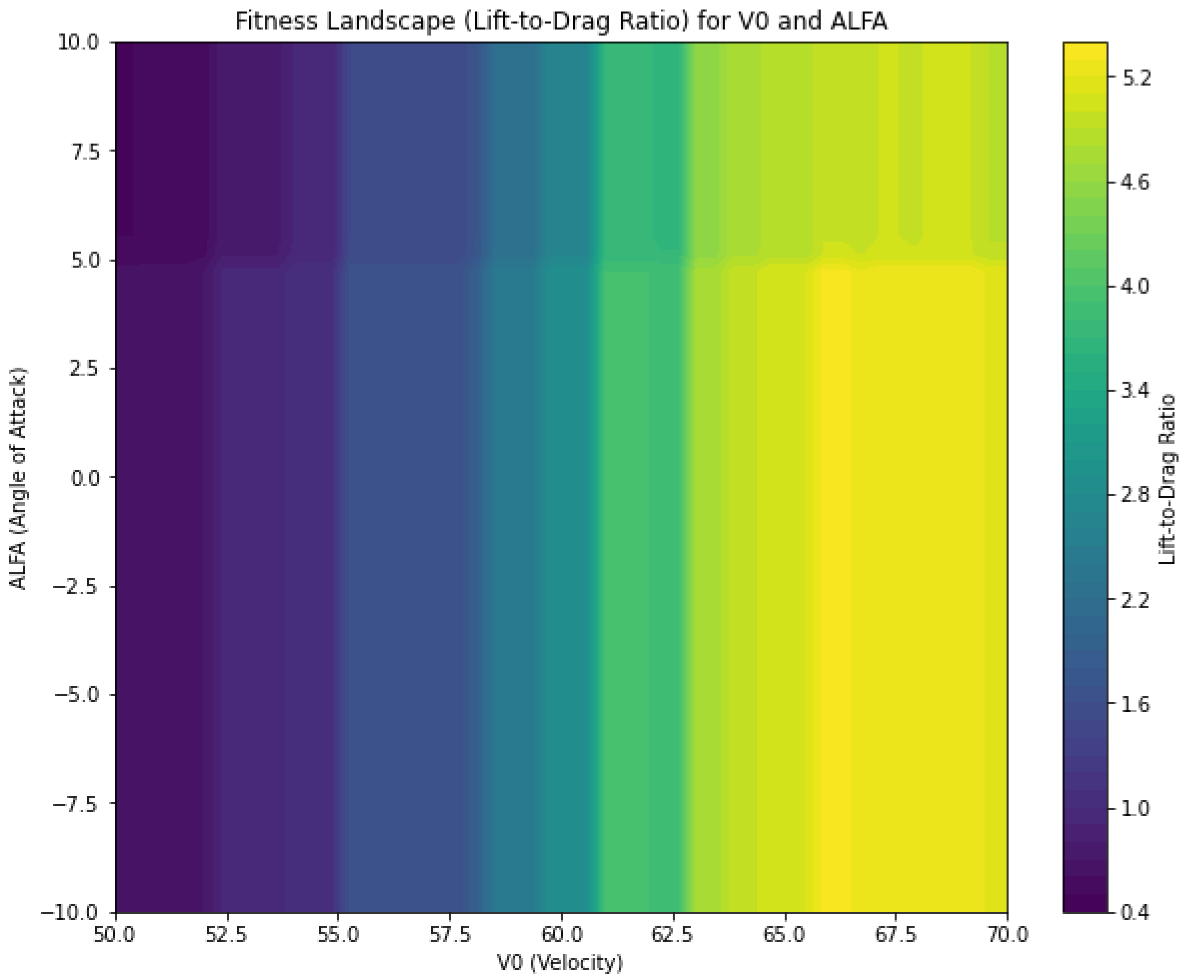
```
CL Model Metrics:
MAE: 0.057670785678915856
MSE: 0.005590352852157927
RMSE: 0.07476866223330418
R²: 0.9917645995317206

CD Model Metrics:
MAE: 0.009010279658290841
MSE: 0.00015997258376996052
RMSE: 0.012648026872597974
R²: 0.9816071516568017
```

Modified GA Using Gradient Boosting

In [19]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
import pygad
import numpy as np
import matplotlib.pyplot as plt
import os

# Step 1: Load all CSV files from the folder and concatenate into a single DataFrame
folder_path = 'Raw Dataset'  # Replace with the folder containing your CSV files
all_data = pd.DataFrame()  # Initialize an empty DataFrame

for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):  # Ensure only CSV files are read
        file_path = os.path.join(folder_path, file_name)
        data = pd.read_csv(file_path)
        all_data = pd.concat([all_data, data], ignore_index=True)

# Step 2: Prepare the data
features = all_data[['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'BETA']]
target_cl = all_data['CL']
target_cd = all_data['CD']

# Step 3: Split the data into training and testing sets for CL and CD
X_train, X_test, y_train_cl, y_test_cl = train_test_split(features, target_cl, test_size
X_train, X_test, y_train_cd, y_test_cd = train_test_split(features, target_cd, test_size

# Step 4: Train Gradient Boosting models for CL and CD
model_cl = GradientBoostingRegressor(n_estimators=100, random_state=42)
model_cl.fit(X_train, y_train_cl)

model_cd = GradientBoostingRegressor(n_estimators=100, random_state=42)
model_cd.fit(X_train, y_train_cd)

# Step 5: Genetic Algorithm (GA) Optimization
best_fitness_values = []  # To store best fitness values

def fitness_function(ga_instance, solution, solution_idx):
    v0, q0, t0, p0, p00, q00, alfa, beta = solution
    feature_set = pd.DataFrame([[v0, q0, t0, p0, p00, q00, alfa, beta]],
                               columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'B
    predicted_cl = model_cl.predict(feature_set)[0]
    predicted_cd = model_cd.predict(feature_set)[0]

    # Prevent division by small CD values
    if predicted_cd < 1e-6:
        predicted_cd = 1e-6

    fitness_value = predicted_cl / predicted_cd  # Maximize Lift-to-Drag ratio
    return fitness_value

# Callback function to store best fitness value after each generation
def on_generation(ga_instance):
    best_fitness = ga_instance.best_solution()[1]
    best_fitness_values.append(best_fitness)

# Adaptive initialization for population
def enhanced_initialization(lower_bounds, upper_bounds, population_size):
    return np.random.uniform(lower_bounds, upper_bounds, (population_size, len(lower_bou

# Adaptive mutation function
def adaptive_mutation(offspring, ga_instance):
```

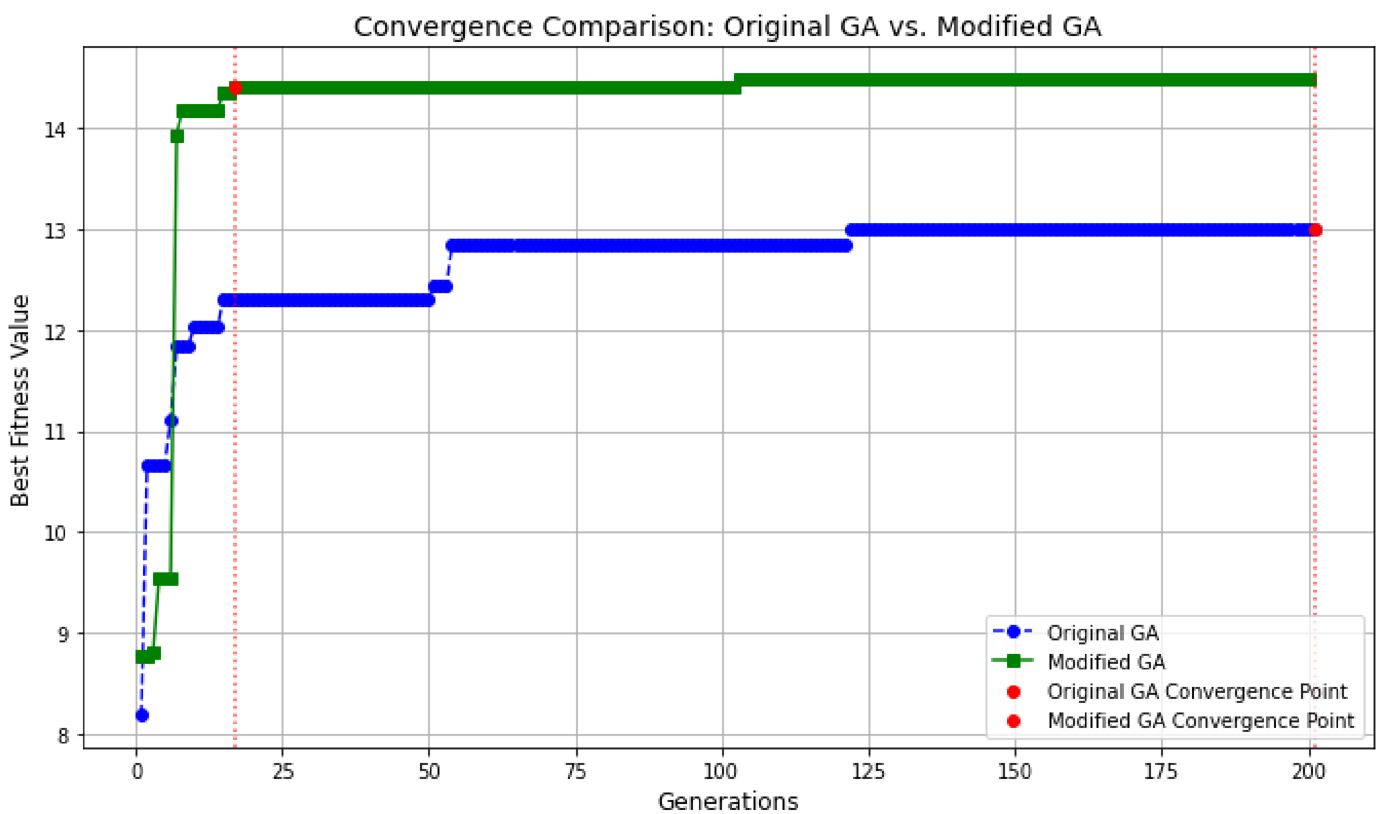
```python
    for idx in range(offspring.shape[0]):
        mutation_indices = np.random.choice(offspring.shape[1], size=2, replace=False)
        for mutation_idx in mutation_indices:
            mutation_amount = np.random.uniform(-0.5, 0.5)  # Larger mutation range
            offspring[idx, mutation_idx] += mutation_amount
    return offspring

# Step 6: Genetic Algorithm Initialization and Execution
lower_bounds = [50, 2000, 300, 100000, 100000, 2000, -10, -5]
upper_bounds = [70, 2500, 350, 101000, 101000, 2500, 10, 5]
population_size = 40

initial_population = enhanced_initialization(lower_bounds, upper_bounds, population_size

ga_instance = pygad.GA(num_generations=200,
                       num_parents_mating=10,
                       fitness_func=fitness_function,
                       initial_population=initial_population,
                       num_genes=8,
                       mutation_type=adaptive_mutation,
                       crossover_probability=0.9,
                       keep_elitism=5,
                       mutation_probability=0.3,
                       on_generation=on_generation)

# Run the Genetic Algorithm
ga_instance.run()

# Step 7: Best Solution
solution, solution_fitness, _ = ga_instance.best_solution()
print("Best Solution (Design Parameters):")
print(f"V0: {solution[0]:.2f}, Q0: {solution[1]:.2f}, T0: {solution[2]:.2f}, "
      f"P0: {solution[3]:.2f}, P00: {solution[4]:.2f}, Q00: {solution[5]:.2f}, "
      f"ALFA: {solution[6]:.2f}, BETA: {solution[7]:.2f}")
print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness:.4f}")

# Step 8: Predicted CL and CD Values
predicted_feature_set = pd.DataFrame([solution], columns=['V0', 'Q0', 'T0', 'P0', 'P00',
predicted_cl = model_cl.predict(predicted_feature_set)[0]
predicted_cd = model_cd.predict(predicted_feature_set)[0]

print(f"Predicted CL value: {predicted_cl:.4f}")
print(f"Predicted CD value: {predicted_cd:.4f}")

# Step 9: Fitness Landscape Visualization
v0_vals = np.linspace(lower_bounds[0], upper_bounds[0], 50)
alfa_vals = np.linspace(lower_bounds[6], upper_bounds[6], 50)
fitness_landscape = np.zeros((50, 50))

for i, v0 in enumerate(v0_vals):
    for j, alfa in enumerate(alfa_vals):
        feature_set = pd.DataFrame([[v0, 2200, 320, 100500, 100500, 2200, alfa, 0]],
                                   columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA'
        predicted_cl = model_cl.predict(feature_set)[0]
        predicted_cd = model_cd.predict(feature_set)[0]
        if predicted_cd < 1e-6:
            predicted_cd = 1e-6
        fitness_landscape[i, j] = predicted_cl / predicted_cd

plt.figure(figsize=(10, 8))
plt.contourf(v0_vals, alfa_vals, fitness_landscape, levels=50, cmap='viridis')
plt.colorbar(label="Lift-to-Drag Ratio")
plt.xlabel("V0 (Velocity)")
plt.ylabel("ALFA (Angle of Attack)")
plt.title("Fitness Landscape (Lift-to-Drag Ratio) for V0 and ALFA")
plt.show()
```

```
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.
3.0. To delay or pause the evolution after each generation, assign a callback function/m
ethod to the 'on_generation' parameter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.
0. To delay or pause the evolution after each generation, assign a callback function/met
hod to the 'on_generation' parameter to adds some time delay.")
Best Solution (Design Parameters):
V0: 55.04, Q0: 2279.76, T0: 303.12, P0: 100669.90, P00: 100317.25, Q00: 2445.08, ALFA:
8.10, BETA: 1.70
Lift-to-Drag Ratio of Best Solution: 14.4928
Predicted CL value: 1.7397
Predicted CD value: 0.1200
```



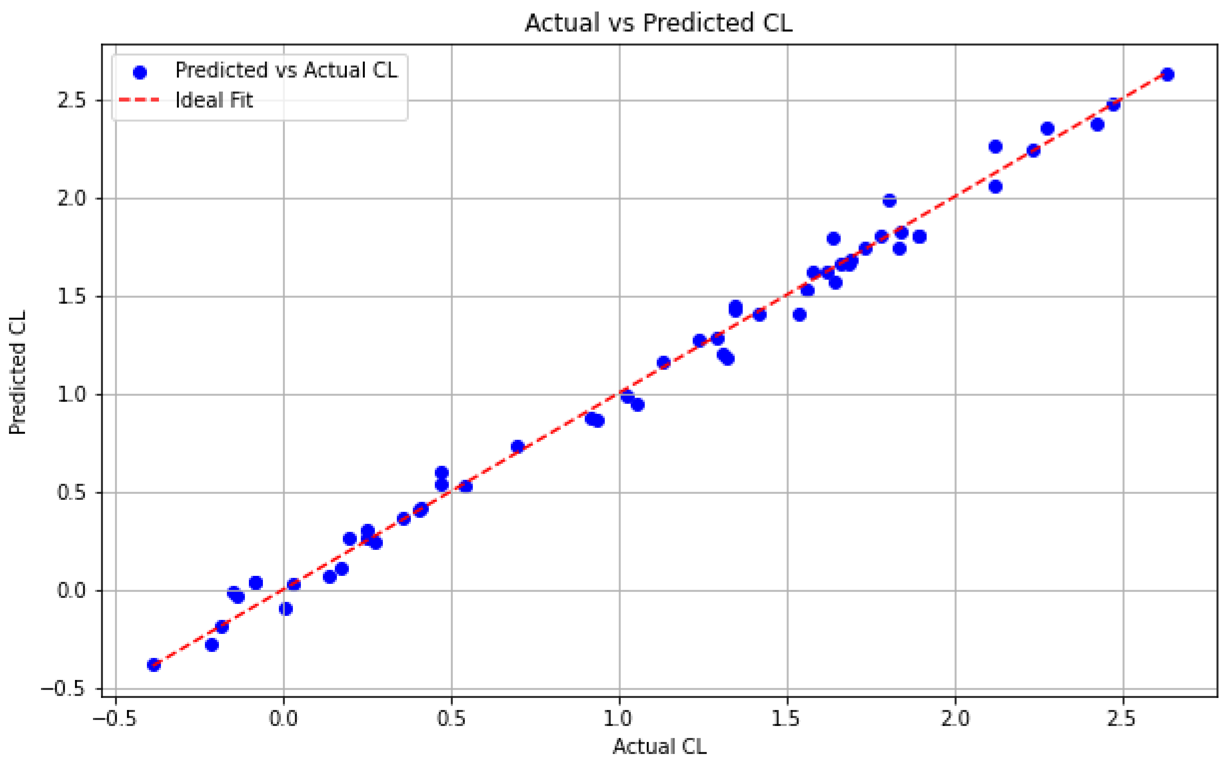
In [20]:
```
import matplotlib.pyplot as plt

# Assuming you have already stored the fitness values for both GAs
# Example: `original_ga_instanceE.fitness_values` and `best_fitness_values` for Modified

# Original GA fitness values (replace with actual fitness data)
original_ga_fitness = ga_instanceE.best_solutions_fitness  # Fitness over generations fr
original_ga_generations = range(1, len(original_ga_fitness) + 1)

# Modified GA fitness values (already stored in `best_fitness_values`)
modified_ga_fitness = best_fitness_values
modified_ga_generations = range(1, len(modified_ga_fitness) + 1)

# Calculate convergence points
original_convergence = len(original_ga_fitness)  # Last generation for Original GA
modified_convergence = next((i + 1 for i, v in enumerate(modified_ga_fitness) if v >= ma

# Get fitness values at convergence points
original_convergence_fitness = original_ga_fitness[-1]  # Fitness at the last generation
```

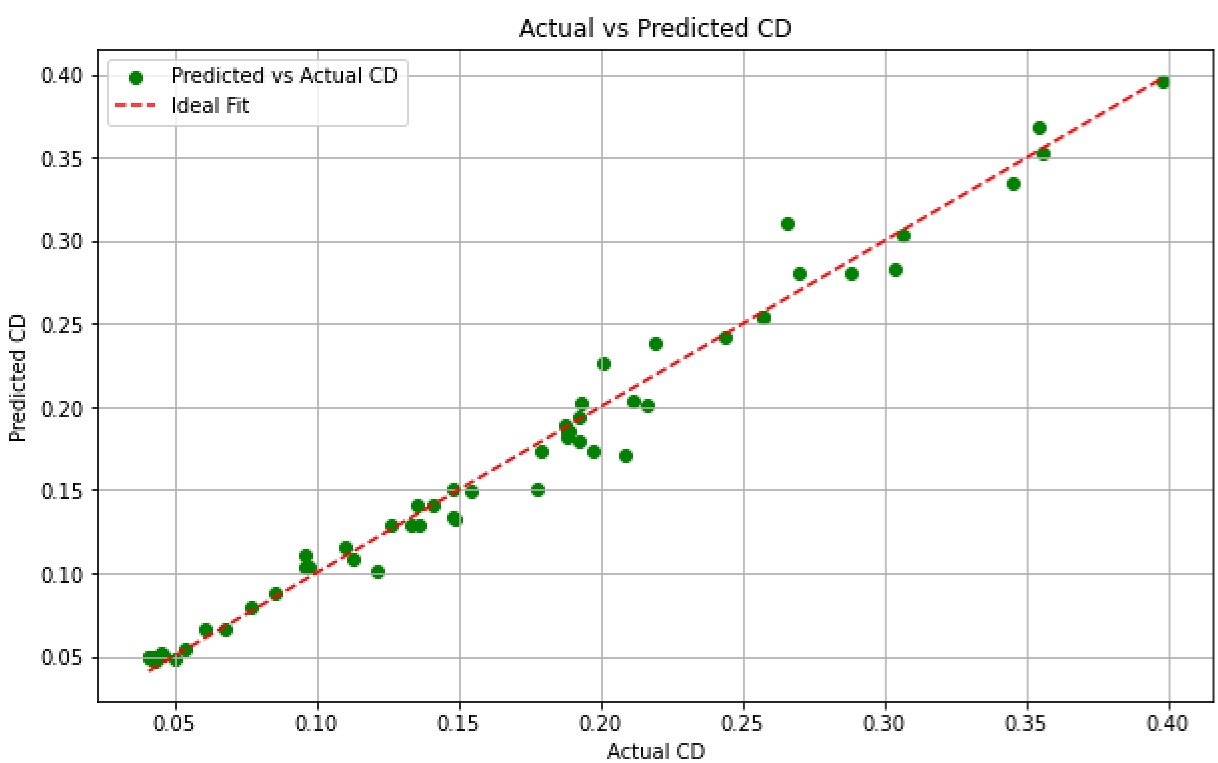
```python
modified_convergence_fitness = modified_ga_fitness[modified_convergence - 1]  # Fitness

# Plot the fitness convergence comparison
plt.figure(figsize=(10, 6))

# Plot for Original GA
plt.plot(original_ga_generations, original_ga_fitness, label="Original GA", color='blue'

# Plot for Modified GA
plt.plot(modified_ga_generations, modified_ga_fitness, label="Modified GA", color='green

# Highlighting convergence points with red markers
plt.scatter(original_convergence, original_convergence_fitness, color='red', label='Orig
plt.scatter(modified_convergence, modified_convergence_fitness, color='red', label='Modi

# Vertical lines for convergence points
plt.axvline(x=original_convergence, color='red', linestyle=':', alpha=0.7)
plt.axvline(x=modified_convergence, color='red', linestyle=':', alpha=0.7)

# Adding labels, title, and legend
plt.title("Convergence Comparison: Original GA vs. Modified GA", fontsize=14)
plt.xlabel("Generations", fontsize=12)
plt.ylabel("Best Fitness Value", fontsize=12)
plt.legend(loc="lower right", fontsize=10)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```



In [21]:
```python
# Step 1: Use the test dataset to make predictions
y_pred_cl = model_cl.predict(X_test)
y_pred_cd = model_cd.predict(X_test)

# Step 2: Plot Actual vs Predicted for CL
plt.figure(figsize=(10, 6))
plt.scatter(y_test_cl, y_pred_cl, color='blue', label='Predicted vs Actual CL')
plt.plot([min(y_test_cl), max(y_test_cl)], [min(y_test_cl), max(y_test_cl)], color='red'
plt.xlabel('Actual CL')
```

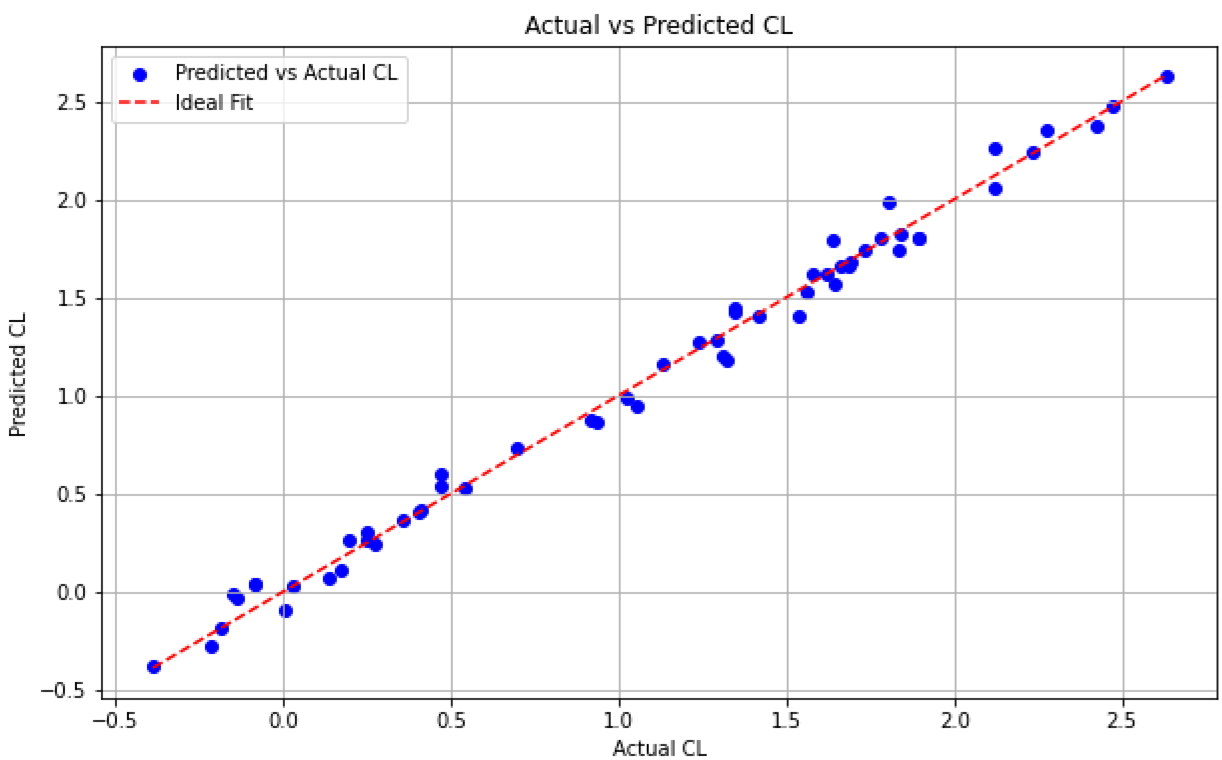
```
plt.ylabel('Predicted CL')
plt.title('Actual vs Predicted CL')
plt.legend()
plt.grid(True)
plt.show()

# Step 3: Plot Actual vs Predicted for CD
plt.figure(figsize=(10, 6))
plt.scatter(y_test_cd, y_pred_cd, color='green', label='Predicted vs Actual CD')
plt.plot([min(y_test_cd), max(y_test_cd)], [min(y_test_cd), max(y_test_cd)], color='red'
plt.xlabel('Actual CD')
plt.ylabel('Predicted CD')
plt.title('Actual vs Predicted CD')
plt.legend()
plt.grid(True)
plt.show()
```

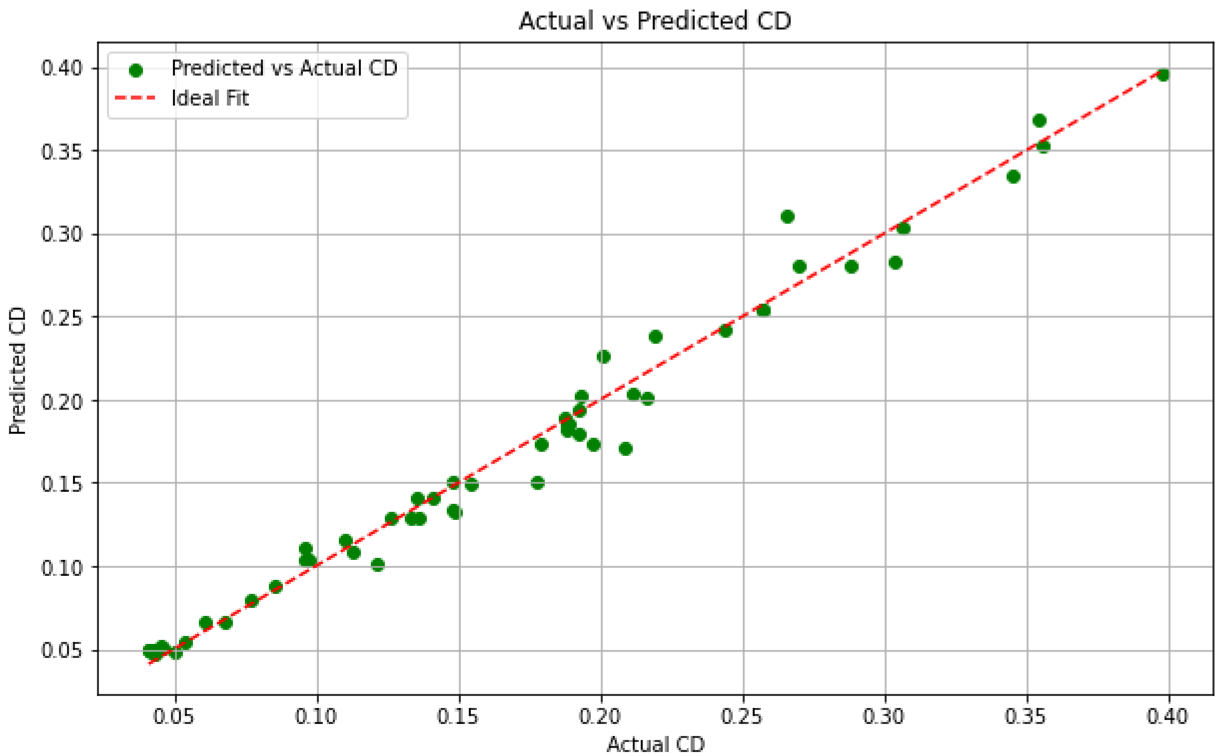




In [22]:

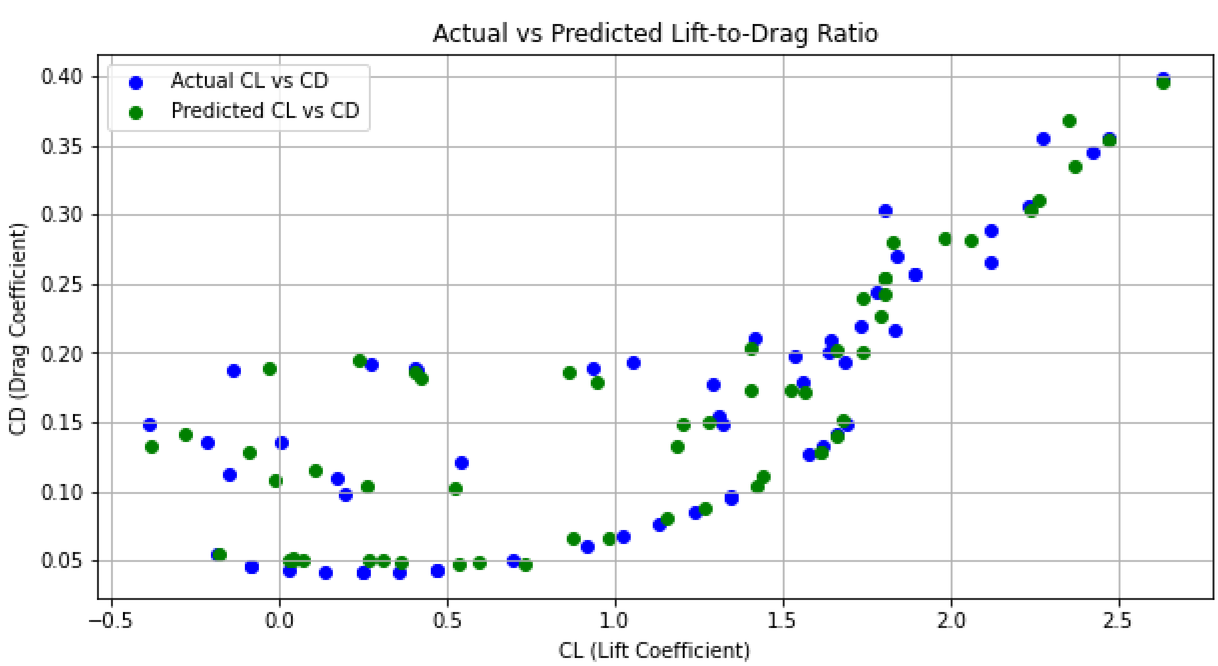
```
# Step 2: Plot Actual vs Predicted for CL
plt.figure(figsize=(10, 6))
plt.scatter(y_test_cl, y_pred_cl, color='blue', label='Predicted vs Actual CL')
plt.plot([min(y_test_cl), max(y_test_cl)], [min(y_test_cl), max(y_test_cl)], color='red'
plt.xlabel('Actual CL')
plt.ylabel('Predicted CL')
plt.title('Actual vs Predicted CL')
plt.legend()
plt.grid(True)
plt.show()

# Step 3: Plot Actual vs Predicted for CD
plt.figure(figsize=(10, 6))
plt.scatter(y_test_cd, y_pred_cd, color='green', label='Predicted vs Actual CD')
plt.plot([min(y_test_cd), max(y_test_cd)], [min(y_test_cd), max(y_test_cd)], color='red'
plt.xlabel('Actual CD')
plt.ylabel('Predicted CD')
plt.title('Actual vs Predicted CD')
plt.legend()
plt.grid(True)
plt.show()
```





In [23]:

```
# Scatter plot for Actual vs Predicted CL and CD
plt.figure(figsize=(10, 5))

# Scatter plot for Actual values
plt.scatter(y_test_cl, y_test_cd, color='blue', label='Actual CL vs CD')

# Scatter plot for Predicted values
predicted_cl_values = model_cl.predict(X_test)
predicted_cd_values = model_cd.predict(X_test)
plt.scatter(predicted_cl_values, predicted_cd_values, color='green', label='Predicted CL

# Adding labels and title
plt.xlabel('CL (Lift Coefficient)')
plt.ylabel('CD (Drag Coefficient)')
plt.title('Actual vs Predicted Lift-to-Drag Ratio')
plt.legend()
plt.grid(True)
plt.show()
```



In [24]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# After training the models, let's calculate the metrics for CL and CD

# Predictions on the test set
y_pred_cl = model_cl.predict(X_test)
y_pred_cd = model_cd.predict(X_test)

# Calculate CL Model Metrics
mae_cl = mean_absolute_error(y_test_cl, y_pred_cl)
mse_cl = mean_squared_error(y_test_cl, y_pred_cl)
rmse_cl = np.sqrt(mse_cl)
r2_cl = r2_score(y_test_cl, y_pred_cl)

print("CL Model Metrics:")
print(f"MAE: {mae_cl}")
print(f"MSE: {mse_cl}")
print(f"RMSE: {rmse_cl}")
print(f"R²: {r2_cl}")

# Calculate CD Model Metrics
mae_cd = mean_absolute_error(y_test_cd, y_pred_cd)
mse_cd = mean_squared_error(y_test_cd, y_pred_cd)
rmse_cd = np.sqrt(mse_cd)
r2_cd = r2_score(y_test_cd, y_pred_cd)

print("\nCD Model Metrics:")
print(f"MAE: {mae_cd}")
print(f"MSE: {mse_cd}")
print(f"RMSE: {rmse_cd}")
print(f"R²: {r2_cd}")
```

```
CL Model Metrics:
MAE: 0.057670785678915856
MSE: 0.005590352852157927
RMSE: 0.07476866223330418
R²: 0.9917645995317206

CD Model Metrics:
MAE: 0.009010279658290841
MSE: 0.00015997258376996052
RMSE: 0.012648026872597974
R²: 0.9816071516568017
```

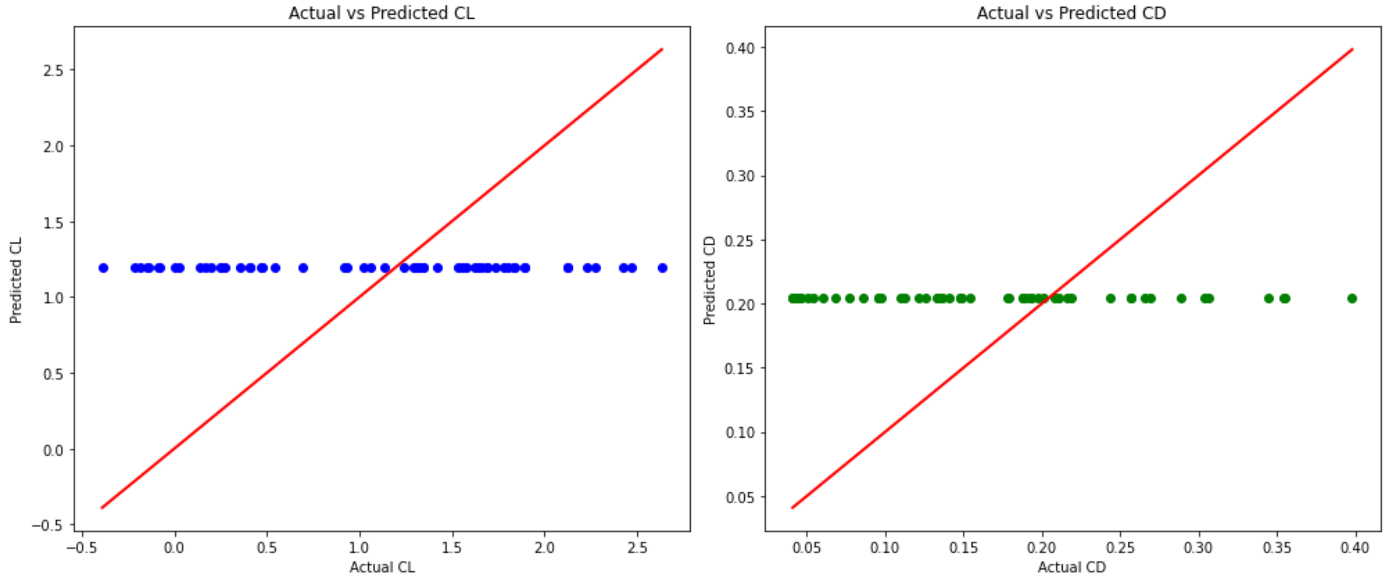
Original GA Using SVM

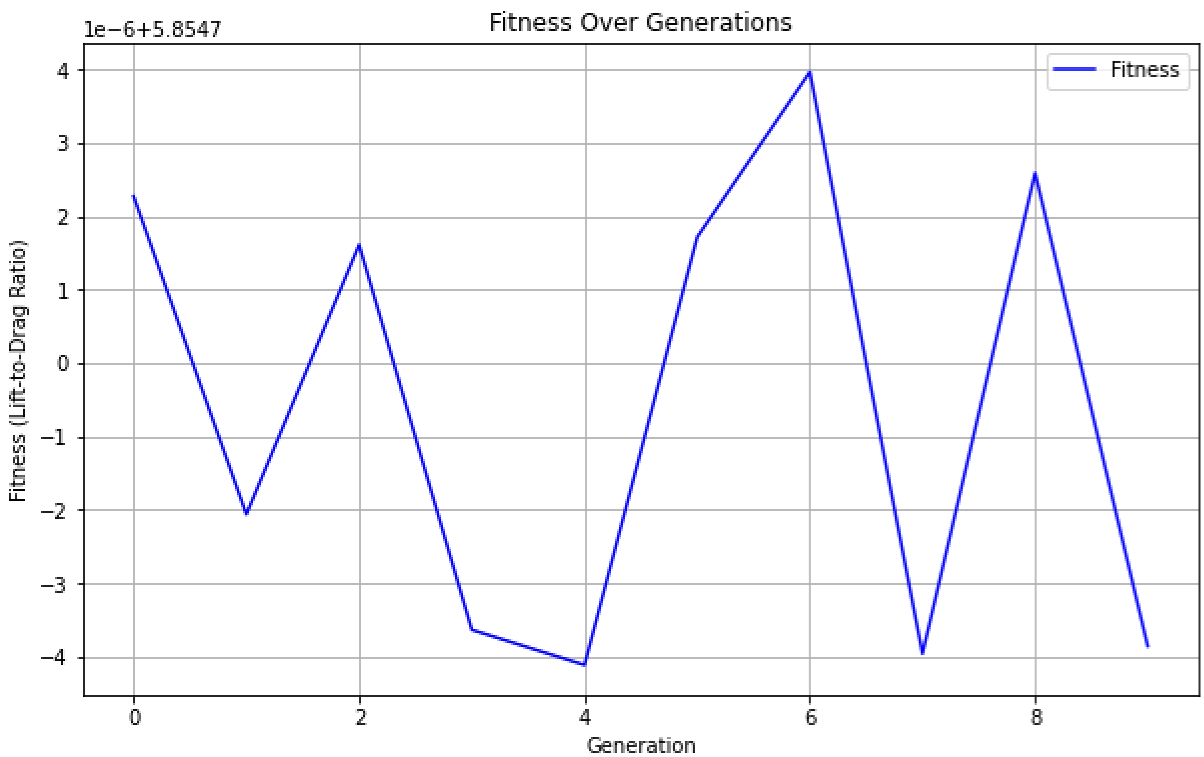
In [25]:
```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
import pygad
import numpy as np
import matplotlib.pyplot as plt

folder_path = 'Raw Dataset'  # Replace with the folder containing your CSV files

# Step 1: Load all CSV files from the folder and concatenate into a single DataFrame
all_data = pd.DataFrame()  # Empty DataFrame to store data from all files

for file_name in os.listdir(folder_path):
    if file_name.endswith('.csv'):  # Ensure only CSV files are read
        file_path = os.path.join(folder_path, file_name)
        data = pd.read_csv(file_path)
```

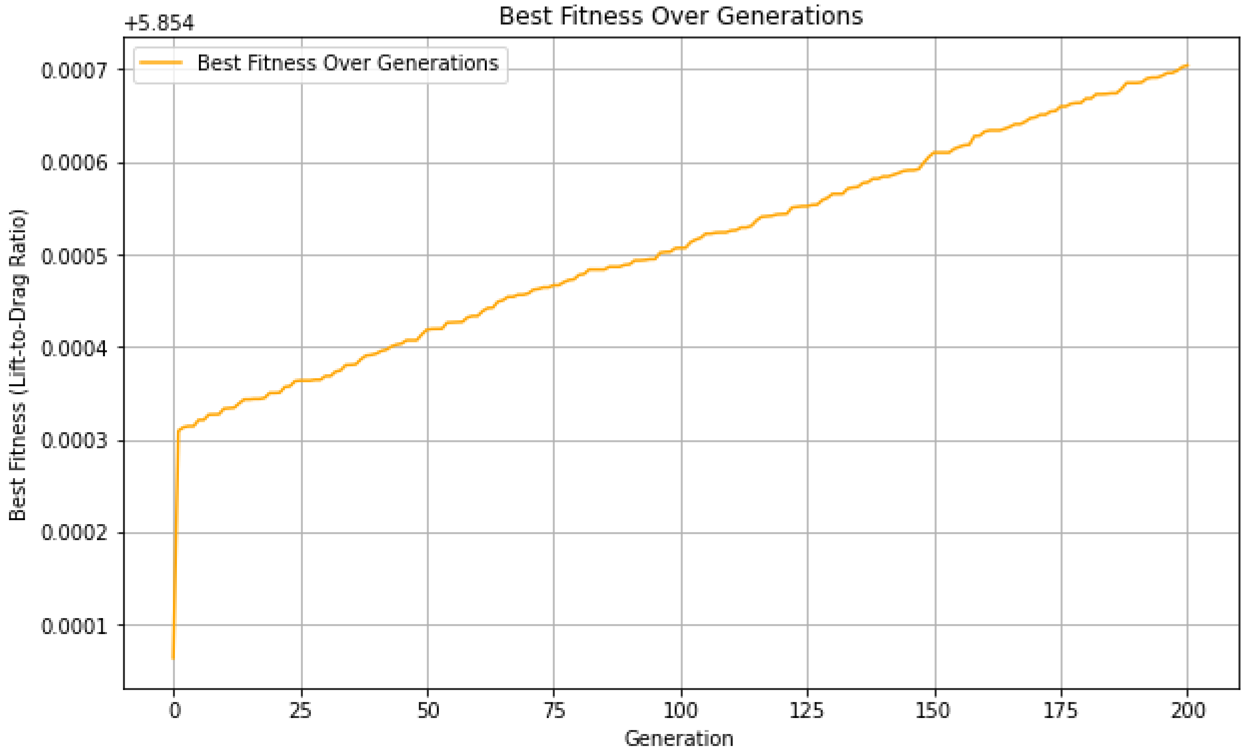
```python
        all_data = pd.concat([all_data, data], ignore_index=True)

# Step 2: Prepare the Data
features = all_data[['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'BETA']]
target_cl = all_data['CL']
target_cd = all_data['CD']

# Step 3: Split Data for Training and Testing
X_train, X_test, y_train_cl, y_test_cl = train_test_split(features, target_cl, test_size
X_train, X_test, y_train_cd, y_test_cd = train_test_split(features, target_cd, test_size

# Step 4: Train the Models (using Support Vector Regressor)
model_cl = SVR(kernel='rbf')  # You can experiment with other kernels like 'linear', 'po
model_cl.fit(X_train, y_train_cl)

model_cd = SVR(kernel='rbf')
model_cd.fit(X_train, y_train_cd)

# Step 5: Define the Genetic Algorithm Fitness Function
def fitness_function(ga_instance, solution, solution_idx):
    v0, q0, t0, p0, p00, q00, alfa, beta = solution
    feature_set = pd.DataFrame([[v0, q0, t0, p0, p00, q00, alfa, beta]],
                               columns=['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', '

    predicted_cl = model_cl.predict(feature_set)[0]
    predicted_cd = model_cd.predict(feature_set)[0]

    if predicted_cd < 1e-6:
        predicted_cd = 1e-6  # Prevent division by zero

    return predicted_cl / predicted_cd  # Maximize Lift-to-Drag ratio

# Step 6: Initialize and Run the Genetic Algorithm
ga_instance = pygad.GA(num_generations=200,
                       num_parents_mating=5,
                       fitness_func=fitness_function,
                       sol_per_pop=10,
                       num_genes=8,
                       init_range_low=[50, 2000, 300, 100000, 100000, 2000, -10, -5],
                       init_range_high=[70, 2500, 350, 101000, 101000, 2500, 10, 5],
                       mutation_percent_genes=10)

# Run the GA
ga_instance.run()

# Step 7: Get and Print the Best Solution
solution, solution_fitness, _ = ga_instance.best_solution()
print(f"Best Solution (Design Parameters): {solution}")
print(f"Lift-to-Drag Ratio of Best Solution: {solution_fitness}")

# Step 8: Predict CL and CD for Test Data
y_pred_cl = model_cl.predict(X_test)
y_pred_cd = model_cd.predict(X_test)

# Step 9: Visualization of CL and CD Predictions vs Actual Values
plt.figure(figsize=(14, 6))

# Plot for CL
plt.subplot(1, 2, 1)
plt.scatter(y_test_cl, y_pred_cl, color='blue')
plt.plot([min(y_test_cl), max(y_test_cl)], [min(y_test_cl), max(y_test_cl)], color='red'
plt.title('Actual vs Predicted CL')
plt.xlabel('Actual CL')
plt.ylabel('Predicted CL')

# Plot for CD
```

```python
plt.subplot(1, 2, 2)
plt.scatter(y_test_cd, y_pred_cd, color='green')
plt.plot([min(y_test_cd), max(y_test_cd)], [min(y_test_cd), max(y_test_cd)], color='red'
plt.title('Actual vs Predicted CD')
plt.xlabel('Actual CD')
plt.ylabel('Predicted CD')

# Show the plots
plt.tight_layout()
plt.show()

# Step 10: Fitness over Generations Visualization
plt.figure(figsize=(10, 6))
plt.plot(ga_instance.last_generation_fitness, color='blue', label='Fitness')
plt.title('Fitness Over Generations')
plt.xlabel('Generation')
plt.ylabel('Fitness (Lift-to-Drag Ratio)')
plt.grid()
plt.legend()
plt.show()

# Additional: Print Best Fitness Value Over Generations
best_fitness_over_generations = ga_instance.best_solutions_fitness
plt.figure(figsize=(10, 6))
plt.plot(best_fitness_over_generations, color='orange', label='Best Fitness Over Generat
plt.title('Best Fitness Over Generations')
plt.xlabel('Generation')
plt.ylabel('Best Fitness (Lift-to-Drag Ratio)')
plt.grid()
plt.legend()
plt.show()
```

```
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
748: UserWarning: The percentage of genes to mutate (mutation_percent_genes=10) resulted
in selecting (0) genes. The number of genes to mutate is set to 1 (mutation_num_genes=
1).
If you do not want to mutate any gene, please set mutation_type=None.
  warnings.warn(f"The percentage of genes to mutate (mutation_percent_genes={mutation_pe
rcent_genes}) resulted in selecting ({mutation_num_genes}) genes. The number of genes to
mutate is set to 1 (mutation_num_genes=1).\nIf you do not want to mutate any gene, pleas
e set mutation_type=None.")
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.
3.0. To delay or pause the evolution after each generation, assign a callback function/m
ethod to the 'on_generation' parameter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.
0. To delay or pause the evolution after each generation, assign a callback function/met
hod to the 'on_generation' parameter to adds some time delay.")
Best Solution (Design Parameters): [ 5.83544606e+01  2.49120785e+03  3.30430869e+02  1.0
0038884e+05
  1.00439941e+05  2.36685501e+03 -1.21272793e+01 -2.15142002e+00]
Lift-to-Drag Ratio of Best Solution: 5.854702267634307
```


Actual vs Predicted CL
Predicted CL
Actual CL
Actual vs Predicted CD
Predicted CD
Actual CD


Fitness Over Generations
1e−6+5.8547
Fitness
Fitness (Lift-to-Drag Ratio)
Generation



Modified GA Using SVM

In [26]:
```
pip install --upgrade pygad
```

```
Requirement already satisfied: pygad in c:\users\legion\appdata\local\programs\python\py
thon38\lib\site-packages (3.3.1)
Requirement already satisfied: cloudpickle in c:\users\legion\appdata\local\programs\pyt
hon\python38\lib\site-packages (from pygad) (3.0.0)
Requirement already satisfied: matplotlib in c:\users\legion\appdata\local\programs\pyth
on\python38\lib\site-packages (from pygad) (3.5.2)
Requirement already satisfied: numpy in c:\users\legion\appdata\local\programs\python\py
thon38\lib\site-packages (from pygad) (1.24.3)
Requirement already satisfied: cycler>=0.10 in c:\users\legion\appdata\local\programs\py
thon\python38\lib\site-packages (from matplotlib->pygad) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\legion\appdata\local\progra
ms\python\python38\lib\site-packages (from matplotlib->pygad) (4.34.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\legion\appdata\local\progra
ms\python\python38\lib\site-packages (from matplotlib->pygad) (1.4.3)
Requirement already satisfied: packaging>=20.0 in c:\users\legion\appdata\local\programs
\python\python38\lib\site-packages (from matplotlib->pygad) (23.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\legion\appdata\local\programs\p
ython\python38\lib\site-packages (from matplotlib->pygad) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\legion\appdata\local\program
s\python\python38\lib\site-packages (from matplotlib->pygad) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\legion\appdata\local\pro
grams\python\python38\lib\site-packages (from matplotlib->pygad) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\legion\appdata\local\programs\python
\python38\lib\site-packages (from python-dateutil>=2.7->matplotlib->pygad) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Error parsing dependencies of bleach: Expected matching RIGHT_PARENTHESIS for L
EFT_PARENTHESIS, after version specifier
    tinycss2 (>=1.1.0<1.2) ; extra == 'css'
             ~~~~~~~~^

[notice] A new release of pip is available: 24.1.1 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [ ]:

In [27]:

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
import pygad
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler

# -------------------------------
# Step 1: Data Loading and Preparation
# -------------------------------

# Modify this directory path to point to your dataset
data_dir = 'Raw Dataset'  # Update this with the correct path

# Initialize an empty DataFrame to hold all data
dataset = pd.DataFrame()

# Iterate through each file in the directory and concatenate CSV files
for fname in os.listdir(data_dir):
    if fname.lower().endswith('.csv'):
        file_path = os.path.join(data_dir, fname)
        temp_df = pd.read_csv(file_path)
        dataset = pd.concat([dataset, temp_df], ignore_index=True)

# Define feature columns and target variables
feature_cols = ['V0', 'Q0', 'T0', 'P0', 'P00', 'Q00', 'ALFA', 'BETA']
target_CL = dataset['CL']
target_CD = dataset['CD']

# -------------------------------
# Step 2: Train-Test Split
# -------------------------------

# Split data for CL prediction
X_train_CL, X_test_CL, y_train_CL, y_test_CL = train_test_split(
    dataset[feature_cols],
    target_CL,
    test_size=0.2,
    random_state=42
)

# Split data for CD prediction
X_train_CD, X_test_CD, y_train_CD, y_test_CD = train_test_split(
    dataset[feature_cols],
    target_CD,
    test_size=0.2,
    random_state=42
)

# -------------------------------
# Step 3: Feature Scaling
# -------------------------------

# Initialize scalers
scaler_CL = StandardScaler()
scaler_CD = StandardScaler()

# Fit scalers on training data and transform both training and test data
X_train_CL_scaled = scaler_CL.fit_transform(X_train_CL)
X_test_CL_scaled = scaler_CL.transform(X_test_CL)
```

```python
X_train_CD_scaled = scaler_CD.fit_transform(X_train_CD)
X_test_CD_scaled = scaler_CD.transform(X_test_CD)

# -------------------------------
# Step 4: Model Training with SVM
# -------------------------------

# Initialize and train SVR model for CL
svm_CL = SVR(kernel='rbf', C=100, epsilon=0.1)
svm_CL.fit(X_train_CL_scaled, y_train_CL)

# Initialize and train SVR model for CD
svm_CD = SVR(kernel='rbf', C=100, epsilon=0.1)
svm_CD.fit(X_train_CD_scaled, y_train_CD)

# -------------------------------
# Step 5: Genetic Algorithm Setup
# -------------------------------

# List to store the best fitness value from each generation
fitness_history = []

# Define the fitness function for GA
def ga_fitness_function(ga_instance, solution, solution_idx):
    # Extract individual genes
    velocity, flow_rate, temperature, pressure1, pressure2, flow_rate2, alpha, beta = so

    # Create a DataFrame for the solution
    input_features = pd.DataFrame([[velocity, flow_rate, temperature, pressure1, pressur

    # Scale the input features
    input_scaled_CL = scaler_CL.transform(input_features)
    input_scaled_CD = scaler_CD.transform(input_features)

    # Predict CL and CD using the trained SVM models
    predicted_CL = svm_CL.predict(input_scaled_CL)[0]
    predicted_CD = svm_CD.predict(input_scaled_CD)[0]

    # Avoid division by zero or very small CD values
    predicted_CD = max(predicted_CD, 1e-6)

    # Calculate Lift-to-Drag ratio
    lift_to_drag = predicted_CL / predicted_CD

    return lift_to_drag

# Callback function to capture the best fitness value each generation
def ga_callback_generation(ga_instance):
    best_fitness = ga_instance.best_solution()[1]
    fitness_history.append(best_fitness)
    # Optional: Print progress every 20 generations
    if (ga_instance.generations_completed % 20) == 0:
        print(f"Generation {ga_instance.generations_completed}: Best Fitness = {best_fit

# Define population boundaries
lower_bounds = [50, 2000, 300, 100000, 100000, 2000, -10, -5]
upper_bounds = [70, 2500, 350, 101000, 101000, 2500, 10, 5]

# Initialize the population using uniform distribution within bounds
def initialize_population(low, high, pop_size, gene_count):
    return np.random.uniform(low, high, (pop_size, gene_count))

initial_population = initialize_population(
    low=lower_bounds,
    high=upper_bounds,
    pop_size=40,        # Increased population size for better diversity
```

```python
    gene_count=8
)

# Define custom crossover and mutation functions
def custom_crossover(parents, offspring_size, ga_instance):
    offspring = np.empty(offspring_size)
    for k in range(offspring_size[0]):
        parent1 = parents[np.random.randint(0, parents.shape[0])]
        parent2 = parents[np.random.randint(0, parents.shape[0])]
        crossover_pt = np.random.randint(1, parents.shape[1])
        offspring[k, 0:crossover_pt] = parent1[0:crossover_pt]
        offspring[k, crossover_pt:] = parent2[crossover_pt:]
    return offspring

def custom_mutation(offspring, ga_instance):
    for idx in range(offspring.shape[0]):
        mutation_indices = np.random.choice(offspring.shape[1], size=2, replace=False)
        for gene_idx in mutation_indices:
            mutation_value = np.random.uniform(-0.5, 0.5)
            offspring[idx, gene_idx] += mutation_value
            offspring[idx, gene_idx] = np.clip(
                offspring[idx, gene_idx],
                lower_bounds[gene_idx],
                upper_bounds[gene_idx]
            )
    return offspring

# Initialize the Genetic Algorithm
ga_instance = pygad.GA(
    num_generations=200,
    num_parents_mating=10,
    fitness_func=ga_fitness_function,
    initial_population=initial_population,
    num_genes=8,
    mutation_type=custom_mutation,
    crossover_type=custom_crossover,
    keep_elitism=5,
    mutation_probability=0.3,
    on_generation=ga_callback_generation
)

# -------------------------------
# Step 6: Run the Genetic Algorithm
# -------------------------------

print("Starting Genetic Algorithm Optimization...")
ga_instance.run()
print("Genetic Algorithm Optimization Completed.")

# Retrieve the best solution found by GA
best_solution, best_fitness, best_solution_idx = ga_instance.best_solution()
print(f"\nBest Solution Parameters:\n{best_solution}")
print(f"Best Lift-to-Drag Ratio: {best_fitness:.4f}")

# -------------------------------
# Step 7: Predictions with the Best Solution
# -------------------------------

# Create a DataFrame for the best solution
best_input = pd.DataFrame([best_solution], columns=feature_cols)

# Scale the input features
best_input_scaled_CL = scaler_CL.transform(best_input)
best_input_scaled_CD = scaler_CD.transform(best_input)

# Predict CL and CD using the best solution
```

```python
best_predicted_CL = svm_CL.predict(best_input_scaled_CL)[0]
best_predicted_CD = svm_CD.predict(best_input_scaled_CD)[0]
best_predicted_CD = max(best_predicted_CD, 1e-6)  # Prevent division by zero

print(f"\nPredicted CL for Best Solution: {best_predicted_CL:.4f}")
print(f"Predicted CD for Best Solution: {best_predicted_CD:.6f}")
print(f"Lift-to-Drag Ratio for Best Solution: {best_predicted_CL / best_predicted_CD:.4f

# -------------------------------
# Step 8: Evaluate Models on Test Data
# -------------------------------

# Predict on test data for CL and CD
test_pred_CL = svm_CL.predict(X_test_CL_scaled)
test_pred_CD = svm_CD.predict(X_test_CD_scaled)

# Calculate Lift-to-Drag ratio for test predictions
test_lift_to_drag = test_pred_CL / np.maximum(test_pred_CD, 1e-6)

# -------------------------------
# Step 9: Visualization
# -------------------------------

# Plot Actual vs Predicted CL
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(y_test_CL, test_pred_CL, color='blue', alpha=0.6, label='Data Points')
plt.plot([y_test_CL.min(), y_test_CL.max()], [y_test_CL.min(), y_test_CL.max()], 'r--',
plt.title('Actual vs Predicted CL')
plt.xlabel('Actual CL')
plt.ylabel('Predicted CL')
plt.legend()
plt.grid(True)

# Plot Actual vs Predicted CD
plt.subplot(1, 2, 2)
plt.scatter(y_test_CD, test_pred_CD, color='green', alpha=0.6, label='Data Points')
plt.plot([y_test_CD.min(), y_test_CD.max()], [y_test_CD.min(), y_test_CD.max()], 'r--',
plt.title('Actual vs Predicted CD')
plt.xlabel('Actual CD')
plt
```

```
Starting Genetic Algorithm Optimization...
```

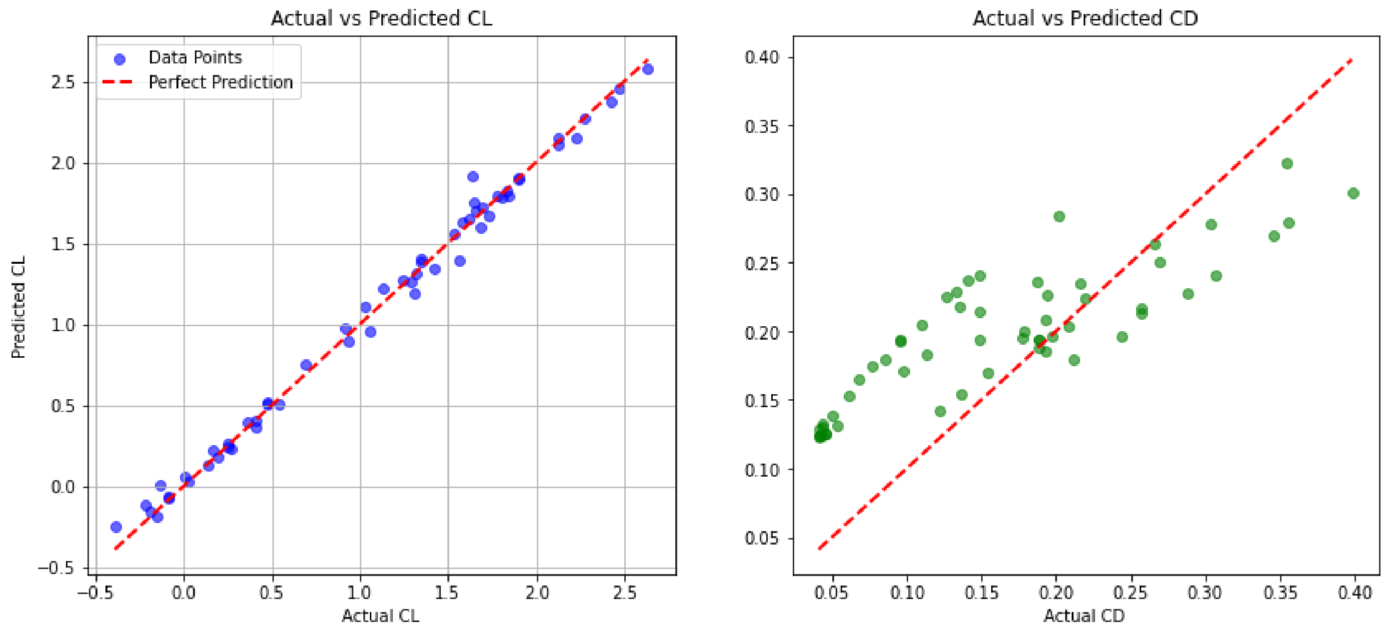
```
C:\Users\LEGION\AppData\Local\Programs\Python\Python38\lib\site-packages\pygad\pygad.py:
1139: UserWarning: The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.
3.0. To delay or pause the evolution after each generation, assign a callback function/m
ethod to the 'on_generation' parameter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.
0. To delay or pause the evolution after each generation, assign a callback function/met
hod to the 'on_generation' parameter to adds some time delay.")
```

```
Generation 20: Best Fitness = 4.6587
Generation 40: Best Fitness = 4.6587
Generation 60: Best Fitness = 4.6587
Generation 80: Best Fitness = 4.6587
Generation 100: Best Fitness = 4.6587
Generation 120: Best Fitness = 4.6587
Generation 140: Best Fitness = 4.6587
Generation 160: Best Fitness = 4.6587
Generation 180: Best Fitness = 4.6587
Generation 200: Best Fitness = 4.6587
Genetic Algorithm Optimization Completed.

Best Solution Parameters:
[ 6.33763363e+01  2.36243110e+03  3.11661812e+02  1.00156764e+05
```

```
  1.00788028e+05  2.41385139e+03 -8.22262256e+00  2.82931303e+00]
Best Lift-to-Drag Ratio: 4.6587

Predicted CL for Best Solution: 1.3106
Predicted CD for Best Solution: 0.281330
Lift-to-Drag Ratio for Best Solution: 4.6587
```

Out[27]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\LEGION\\AppData\\Local\\Programs\\Python\\Python38\\lib\\site-packages\\matplotlib\\pyplot.py'>
```



In [28]:

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# -------------------------------
# Step 8: Evaluate Models on Test Data
# -------------------------------

# Predict on test data for CL and CD
test_pred_CL = svm_CL.predict(X_test_CL_scaled)
test_pred_CD = svm_CD.predict(X_test_CD_scaled)

# Calculate Lift-to-Drag ratio for test predictions
test_lift_to_drag = test_pred_CL / np.maximum(test_pred_CD, 1e-6)

# -------------------------------
# Step 10: Calculate Evaluation Metrics
# -------------------------------

# Metrics for CL
mae_CL = mean_absolute_error(y_test_CL, test_pred_CL)
mse_CL = mean_squared_error(y_test_CL, test_pred_CL)
rmse_CL = np.sqrt(mse_CL)
r2_CL = r2_score(y_test_CL, test_pred_CL)

# Metrics for CD
mae_CD = mean_absolute_error(y_test_CD, test_pred_CD)
mse_CD = mean_squared_error(y_test_CD, test_pred_CD)
rmse_CD = np.sqrt(mse_CD)
r2_CD = r2_score(y_test_CD, test_pred_CD)

# Display metrics
print("\nCL Model Metrics:")
print(f"MAE: {mae_CL:.6f}")
print(f"MSE: {mse_CL:.6f}")
print(f"RMSE: {rmse_CL:.6f}")
print(f"R²: {r2_CL:.6f}")
```

```
print("\nCD Model Metrics:")
print(f"MAE: {mae_CD:.6f}")
print(f"MSE: {mse_CD:.6f}")
print(f"RMSE: {rmse_CD:.6f}")
print(f"R²: {r2_CD:.6f}")
```

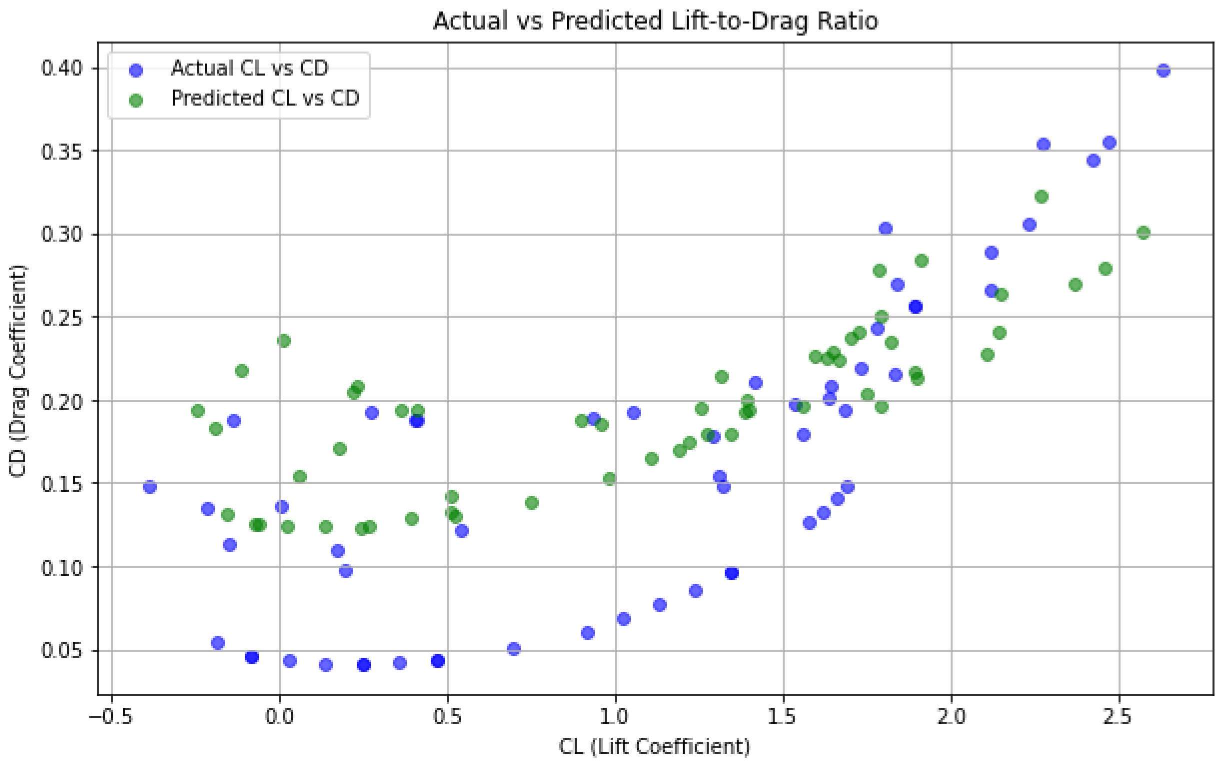
```
CL Model Metrics:
MAE: 0.050785
MSE: 0.004908
RMSE: 0.070055
R²: 0.992770

CD Model Metrics:
MAE: 0.057006
MSE: 0.004419
RMSE: 0.066477
R²: 0.491905
```

In [29]:
```
# -------------------------------
# Step 11: Print Best Solution Parameters in Desired Format
# -------------------------------

# Format the best solution parameters
formatted_params = ", ".join([f"{param}: {value:.2f}" for param, value in zip(feature_co

# Print the best solution parameters in the desired format
print("\nBest Solution (Design Parameters):")
print(formatted_params)

# Display metrics for the best solution
print("\nMetrics for Best Solution:")
print(f"Predicted CL: {best_predicted_CL:.6f}")
print(f"Predicted CD: {best_predicted_CD:.6f}")
print(f"Lift-to-Drag Ratio: {best_predicted_CL / best_predicted_CD:.6f}")
```

```
Best Solution (Design Parameters):
V0: 63.38, Q0: 2362.43, T0: 311.66, P0: 100156.76, P00: 100788.03, Q00: 2413.85, ALFA: -
8.22, BETA: 2.83

Metrics for Best Solution:
Predicted CL: 1.310635
Predicted CD: 0.281330
Lift-to-Drag Ratio: 4.658711
```

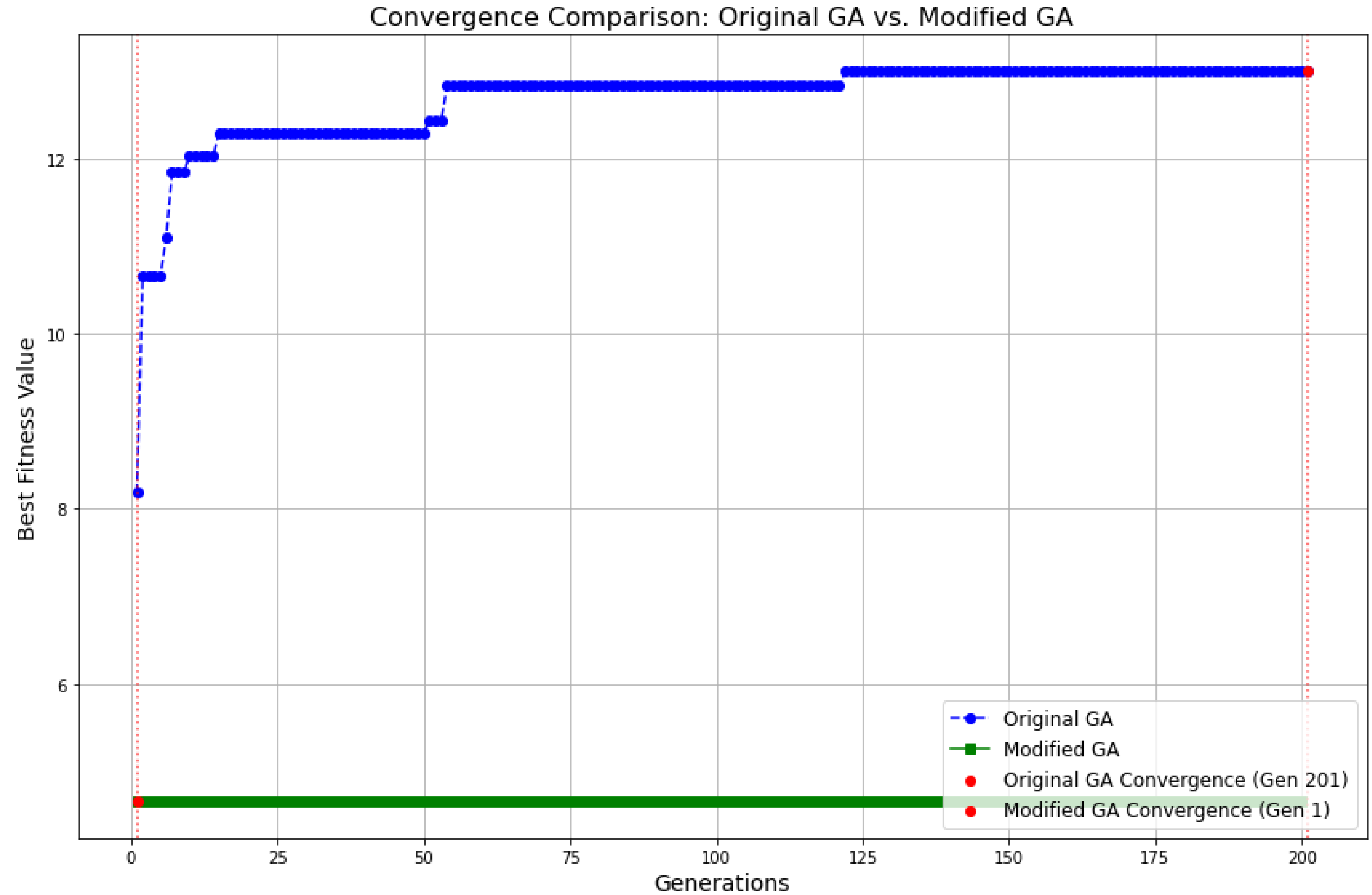
In [30]:
```
import matplotlib.pyplot as plt

# Assuming you have test_pred_CL, test_pred_CD (predicted values) and y_test_CL, y_test_

# Plot CL vs CD for Actual and Predicted values
plt.figure(figsize=(10, 6))

# Actual CL vs CD (blue points)
plt.scatter(y_test_CL, y_test_CD, color='blue', alpha=0.6, label='Actual CL vs CD')

# Predicted CL vs CD (green points)
plt.scatter(test_pred_CL, test_pred_CD, color='green', alpha=0.6, label='Predicted CL vs

# Add labels and title
plt.title('Actual vs Predicted Lift-to-Drag Ratio')
plt.xlabel('CL (Lift Coefficient)')
plt.ylabel('CD (Drag Coefficient)')

# Display the legend
plt.legend()
```

```python
# Show grid
plt.grid(True)

# Display the plot
plt.show()
```



In [31]:
```python
import matplotlib.pyplot as plt

# Assuming you have already stored the fitness values for both GAs
# Replace these with actual data from your GA runs

# Fitness values for the Original GA
original_ga_fitness = ga_instanceE.best_solutions_fitness  # Replace with the actual fit
original_ga_generations = range(1, len(original_ga_fitness) + 1)

# Fitness values for the Modified GA
modified_ga_fitness = fitness_history  # Replace with the fitness history of the Modifie
modified_ga_generations = range(1, len(modified_ga_fitness) + 1)

# Convergence points
original_convergence = len(original_ga_fitness)  # Last generation for the Original GA
modified_convergence = next(
    (i + 1 for i, v in enumerate(modified_ga_fitness) if v >= max(modified_ga_fitness) *
    len(modified_ga_fitness)
)

# Fitness values at convergence points
original_convergence_fitness = original_ga_fitness[-1]  # Fitness at the last generation
modified_convergence_fitness = modified_ga_fitness[modified_convergence - 1]  # Fitness

# Plot the convergence comparison
plt.figure(figsize=(12, 8))

# Original GA fitness plot
plt.plot(
    original_ga_generations,
    original_ga_fitness,
    label="Original GA",
    color='blue',
    linestyle='--',
```

```
    marker='o'
)

# Modified GA fitness plot
plt.plot(
    modified_ga_generations,
    modified_ga_fitness,
    label="Modified GA",
    color='green',
    linestyle='-',
    marker='s'
)

# Mark convergence points with red markers
plt.scatter(
    original_convergence,
    original_convergence_fitness,
    color='red',
    label=f"Original GA Convergence (Gen {original_convergence})",
    zorder=5
)
plt.scatter(
    modified_convergence,
    modified_convergence_fitness,
    color='red',
    label=f"Modified GA Convergence (Gen {modified_convergence})",
    zorder=5
)

# Add vertical lines for convergence points
plt.axvline(
    x=original_convergence,
    color='red',
    linestyle=':',
    alpha=0.7
)
plt.axvline(
    x=modified_convergence,
    color='red',
    linestyle=':',
    alpha=0.7
)

# Add labels, title, and legend
plt.title("Convergence Comparison: Original GA vs. Modified GA", fontsize=16)
plt.xlabel("Generations", fontsize=14)
plt.ylabel("Best Fitness Value", fontsize=14)
plt.legend(loc="lower right", fontsize=12)
plt.grid(True)
plt.tight_layout()

# Show the plot
plt.show()
```


Convergence Comparison: Original GA vs. Modified GA
Best Fitness Value
Generations
Original GA
Modified GA
Original GA Convergence (Gen 201)
Modified GA Convergence (Gen 1)
0
25
50
75
100
125
150
175
200
6
8
10
12