```python
import cupy as cp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.spatial.distance import cdist
from functools import partial
from joblib import Parallel, delayed

# Himmelblau function and its gradient
def himmelblau_cupy(x):
    """Himmelblau function for optimization."""
    return (x[0]**2 + x[1] - 11)**2 + (x[0] + x[1]**2 - 7)**2

def himmelblau_grad_cupy(x):
    """Gradient of the Himmelblau function."""
    grad = cp.zeros(2)
    grad[0] = 4 * x[0] * (x[0]**2 + x[1] - 11) + 2 * (x[0] + x[1]**2 -
7)
    grad[1] = 2 * (x[0]**2 + x[1] - 11) + 4 * x[1] * (x[0] + x[1]**2 -
7)
    return grad

# Known global minima of Himmelblau function
global_minima = np.array([
    [3.0, 2.0],
    [-2.805118, 3.131312],
    [-3.779310, -3.283186],
    [3.584428, -1.848126]
])

# Simulated Annealing (optimized)
def simulated_annealing_cupy(func, x0, T0, alpha, sigma, max_iter,
gpu_id=0):
    with cp.cuda.Device(gpu_id):
        x = cp.array(x0, dtype=cp.float32)
        current_loss = func(x)
        best_x = x.copy()
        best_loss = current_loss
        path = [x.copy()]
        losses = [float(current_loss)]
        T = T0

        for _ in range(max_iter):
            x_new = x + cp.random.normal(0, sigma, x.shape)
            loss_new = func(x_new)
            if loss_new < current_loss or cp.exp((current_loss -
loss_new) / T) > cp.random.rand():
                x = x_new
                current_loss = loss_new
                if loss_new < best_loss:
```

```
                    best_x = x_new
                    best_loss = loss_new
            path.append(x.copy())
            losses.append(float(current_loss))
            T *= alpha

        return cp.array(path), losses

# Adam optimizer
def adam_cupy(func, grad_func, x0, lr=0.001, beta1=0.9, beta2=0.999,
epsilon=1e-8, max_iter=1000, gpu_id=0):
    with cp.cuda.Device(gpu_id):
        x = cp.array(x0, dtype=cp.float32)
        m = cp.zeros_like(x)
        v = cp.zeros_like(x)
        path = [x.copy()]
        losses = [float(func(x))]
        t = 0

        for _ in range(max_iter):
            t += 1
            g = grad_func(x)
            m = beta1 * m + (1 - beta1) * g
            v = beta2 * v + (1 - beta2) * (g**2)
            m_hat = m / (1 - beta1**t)
            v_hat = v / (1 - beta2**t)
            x = x - lr * m_hat / (cp.sqrt(v_hat) + epsilon)
            path.append(x.copy())
            losses.append(float(func(x)))

        return cp.array(path), losses

# Azure_Sky optimizer
def azure_sky(func, grad_func, x0, sa_iter=100, adam_iter=900,
T0=10.0, alpha=0.99, sigma=0.5, lr=0.001, gpu_id=0):
    """Hybrid SA followed by Adam optimizer."""
    sa_path, sa_losses = simulated_annealing_cupy(func, x0, T0, alpha,
sigma, sa_iter, gpu_id=gpu_id)
    best_x = sa_path[-1]
    adam_path, adam_losses = adam_cupy(func, grad_func, best_x, lr=lr,
max_iter=adam_iter, gpu_id=gpu_id)
    with cp.cuda.Device(gpu_id):
        full_path = cp.concatenate([sa_path, adam_path[1:]], axis=0)
    full_losses = sa_losses + adam_losses[1:]
    return full_path, full_losses

# SGD optimizer
def sgd_cupy(func, grad_func, x0, lr=0.01, max_iter=1000, gpu_id=0):
    with cp.cuda.Device(gpu_id):
        x = cp.array(x0, dtype=cp.float32)
```

```
        path = [x.copy()]
        losses = [float(func(x))]

        for _ in range(max_iter):
            g = grad_func(x)
            x = x - lr * g
            path.append(x.copy())
            losses.append(float(func(x)))

        return cp.array(path), losses

# RMSprop optimizer
def rmsprop_cupy(func, grad_func, x0, lr=0.001, gamma=0.9, epsilon=1e-
8, max_iter=1000, gpu_id=0):
    with cp.cuda.Device(gpu_id):
        x = cp.array(x0, dtype=cp.float32)
        Eg2 = cp.zeros_like(x)
        path = [x.copy()]
        losses = [float(func(x))]

        for _ in range(max_iter):
            g = grad_func(x)
            Eg2 = gamma * Eg2 + (1 - gamma) * (g**2)
            x = x - lr * g / (cp.sqrt(Eg2) + epsilon)
            path.append(x.copy())
            losses.append(float(func(x)))

        return cp.array(path), losses

# Compute confidence intervals
def compute_confidence_intervals(data, confidence=0.95):
    mean = np.mean(data)  # Data is a list of Python floats
    std = np.std(data, ddof=1)
    n = len(data)
    margin = 1.96 * std / np.sqrt(n)  # 95% CI
    return f"{mean:.2f} ± {margin:.2f}"

# Generate visualizations
def generate_visualizations(results):
    # Boxplots
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

    steps_data = [results[name]['steps'] for name in results]
    ax1.boxplot(steps_data, labels=results.keys())
    ax1.set_title('Steps to Convergence')
    ax1.set_ylabel('Steps')

    loss_data = [results[name]['final_losses'] for name in results]
    ax2.boxplot(loss_data, labels=results.keys())
    ax2.set_title('Final Loss')
```

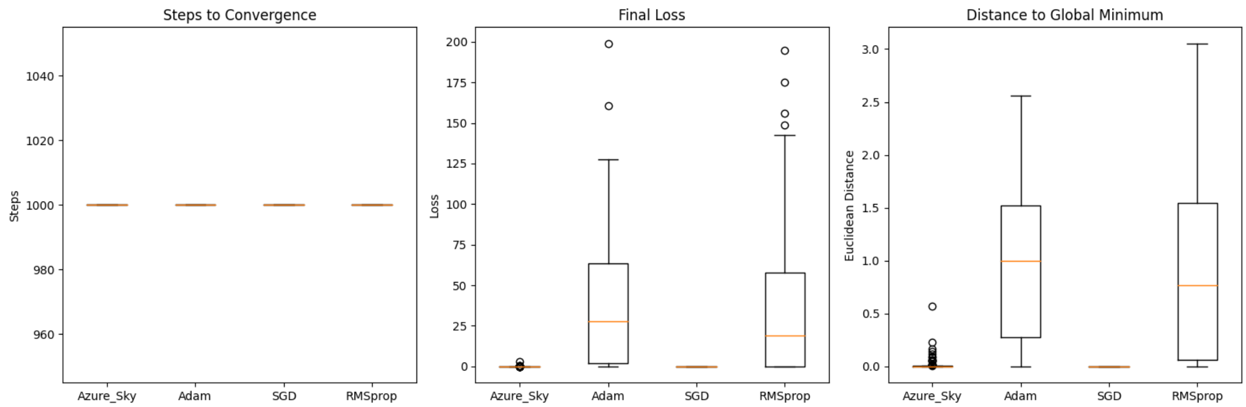
```
    ax2.set_ylabel('Loss')

    dist_data = [results[name]['distances'] for name in results]
    ax3.boxplot(dist_data, labels=results.keys())
    ax3.set_title('Distance to Global Minimum')
    ax3.set_ylabel('Euclidean Distance')

    plt.tight_layout()
    plt.savefig('boxplots.png')
    plt.show()
    plt.close()

    # Convergence trajectories
    plt.figure(figsize=(8, 6))
    for name, data in results.items():
        all_trajectories = np.array([np.array(traj) for traj in
data['loss_trajectories']])
        mean_loss = np.mean(all_trajectories, axis=0)
        plt.plot(mean_loss, label=name)
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title('Convergence Trajectories (Mean Loss)')
    plt.legend()
    plt.grid(True)
    plt.savefig('convergence.png')
    plt.show()
    plt.close()

    # Heatmap of failed runs
    failed_data = {}
    for name in results:
        failed_runs = [(l, d) for l, d, s in zip(results[name]
['final_losses'], results[name]['distances'], results[name]
['success']) if s == 0]
        if failed_runs:
            losses, distances = zip(*failed_runs)
            failed_data[name] = (losses, distances)

    if failed_data:
        fig, ax = plt.subplots(figsize=(8, 6))
        for name, (losses, distances) in failed_data.items():
            hist, xedges, yedges = np.histogram2d(losses, distances,
bins=10, range=[[0, 10], [3.5, 5.5]])
            ax.imshow(hist.T, origin='lower', extent=[xedges[0],
xedges[-1], yedges[0], yedges[-1]], cmap='hot', alpha=0.5, label=name)
        ax.set_xlabel('Final Loss')
        ax.set_ylabel('Distance to Global Minimum')
        ax.set_title('Heatmap of Failed Runs')
        plt.legend()
```

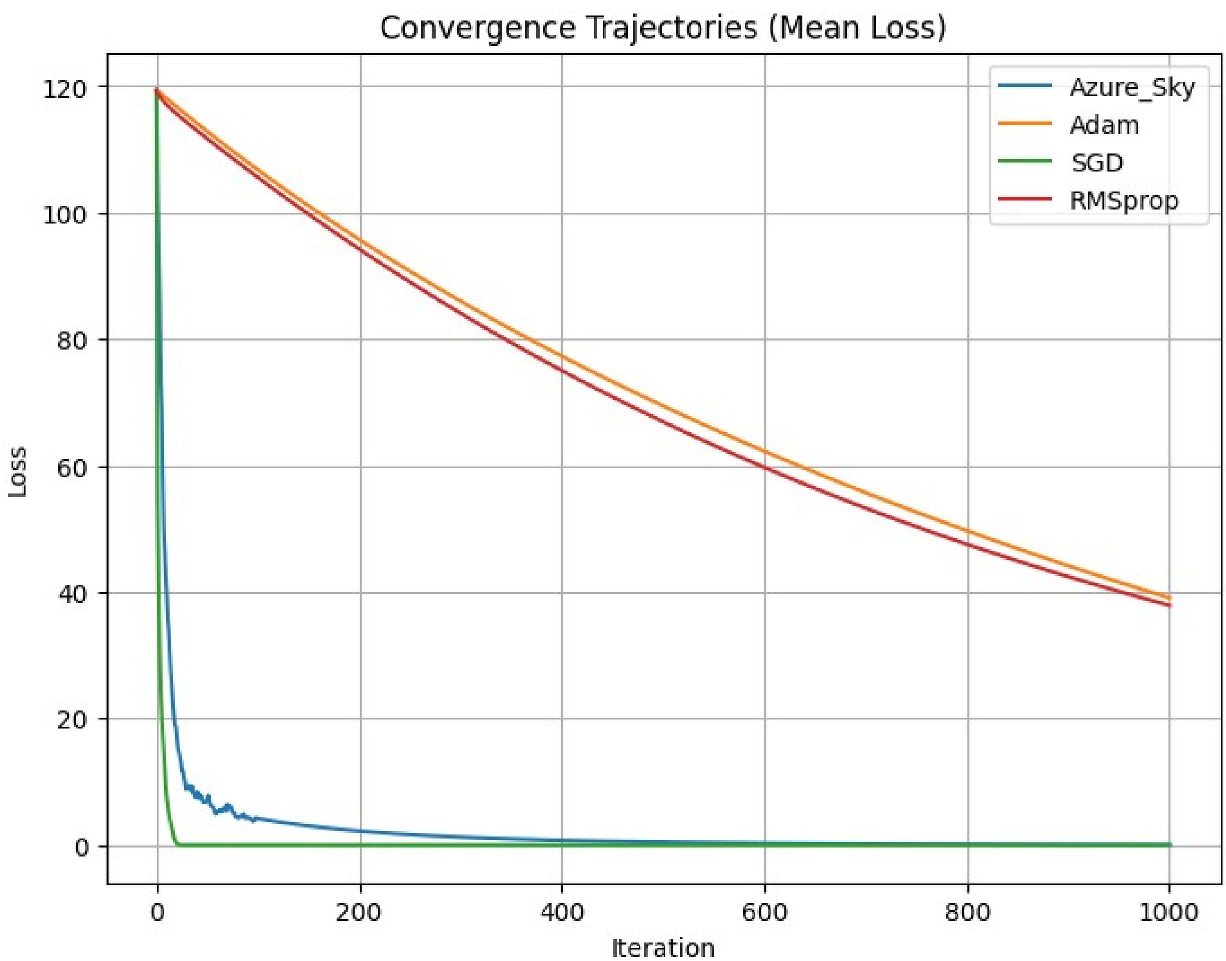
```
        plt.savefig('heatmap.png')
        plt.show()
        plt.close()

# Run single optimization
def run_single_opt(opt_func, x0, gpu_id):
    with cp.cuda.Device(gpu_id):
        path, losses = opt_func(x0)
        steps = len(losses) - 1
        final_loss = float(losses[-1])  # Convert to Python float
        final_pos = cp.asnumpy(path[-1])  # Convert to NumPy
        min_dist = float(np.min(cdist([final_pos], global_minima,
metric='euclidean')))  # Convert to Python float
        success = 1 if final_loss < 1.0 else 0
        return steps, final_loss, min_dist, success, losses

# Benchmark function with GPU load distribution
def run_benchmark(n_runs=100, max_iter=1000):
    n_gpus = cp.cuda.runtime.getDeviceCount()  # Detect number of GPUs
(2 T4s)

    optimizers = {
        'Azure_Sky': partial(azure_sky, himmelblau_cupy,
himmelblau_grad_cupy, sa_iter=100, adam_iter=900, T0=10.0, alpha=0.99,
sigma=0.5, lr=0.001),
        'Adam': partial(adam_cupy, himmelblau_cupy,
himmelblau_grad_cupy, lr=0.001, max_iter=max_iter),
        'SGD': partial(sgd_cupy, himmelblau_cupy,
himmelblau_grad_cupy, lr=0.01, max_iter=max_iter),
        'RMSprop': partial(rmsprop_cupy, himmelblau_cupy,
himmelblau_grad_cupy, lr=0.001, max_iter=max_iter)
    }

    results = {name: {'steps': [], 'final_losses': [], 'distances':
[], 'success': [], 'loss_trajectories': []} for name in optimizers}

    initial_points = [np.random.uniform(-5, 5, 2) for _ in
range(n_runs)]

    for name, opt_func in optimizers.items():
        opt_results = Parallel(n_jobs=4)(delayed(run_single_opt)
(opt_func, x0, i % n_gpus) for i, x0 in enumerate(initial_points))
        for res in opt_results:
            steps, final_loss, min_dist, success, losses = res
            results[name]['steps'].append(steps)
            results[name]['final_losses'].append(final_loss)
            results[name]['distances'].append(min_dist)
            results[name]['success'].append(success)
            results[name]['loss_trajectories'].append(losses)
```

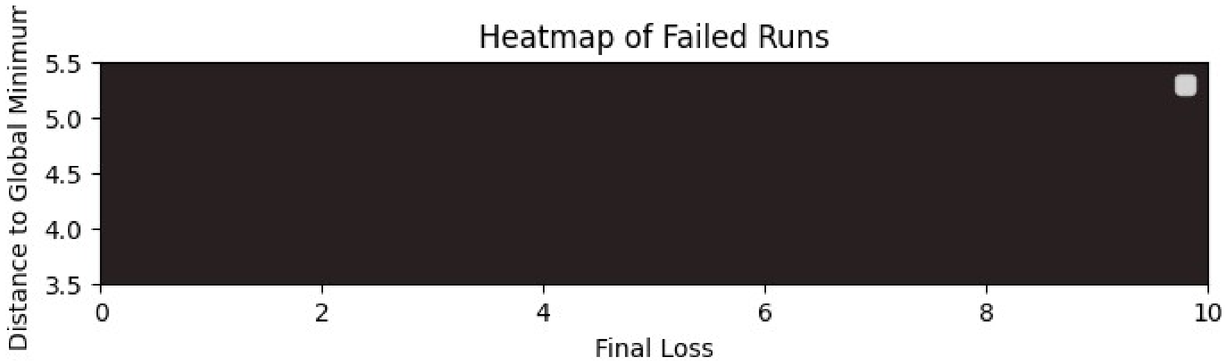
```
    return results

# Main execution
results = run_benchmark(n_runs=100, max_iter=1000)

# Compute summary statistics
summary = {
    'Optimizer': [],
    'Avg Steps': [],
    'Avg Loss': [],
    'Success Rate (%)': [],
    'CI Steps': [],
    'CI Loss': [],
    'CI Distance': []
}
for name in results:
    summary['Optimizer'].append(name)
    summary['Avg Steps'].append(np.mean(results[name]['steps']))
    summary['Avg Loss'].append(np.mean(results[name]['final_losses']))
    summary['Success Rate (%)'].append(100 * np.mean(results[name]
['success']))
    summary['CI
Steps'].append(compute_confidence_intervals(results[name]['steps']))
    summary['CI
Loss'].append(compute_confidence_intervals(results[name]
['final_losses']))
    summary['CI
Distance'].append(compute_confidence_intervals(results[name]
['distances']))

# Save and display summary
df = pd.DataFrame(summary)
df.to_csv('optimizer_summary.csv', index=False)
print(df)

# Generate visualizations
generate_visualizations(results)
```

```
   Optimizer  Avg Steps      Avg Loss  Success Rate (%)        CI
Steps  \
0  Azure_Sky     1000.0  5.320019e-02              99.0  1000.00 ±
0.00
1       Adam     1000.0  3.906899e+01              20.0  1000.00 ±
0.00
2        SGD     1000.0  3.762867e-30             100.0  1000.00 ±
0.00
3    RMSprop     1000.0  3.788616e+01              30.0  1000.00 ±
0.00
```

	CI Loss	CI Distance
0	0.05 ± 0.07	0.02 ± 0.01
1	39.07 ± 8.29	0.98 ± 0.14
2	0.00 ± 0.00	0.00 ± 0.00
3	37.89 ± 9.22	0.96 ± 0.18







Heatmap of Failed Runs
Distance to Global Minimum
5.5
5.0
4.5
4.0
3.5
0
2
4
6
8
10
Final Loss