Configuration Manual

MSc Research Project
Cloud Computing

Aryan Kumar
Student ID: 23310618

School of Computing
National College of Ireland

Supervisor: Prof. Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aryan Kumar
Student ID:	23310618
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Prof. Punit Gupta
Submission Due Date:	15/09/2025
Project Title:	Hybrid Bacterial Colony Optimization and Particle Swarm Optimization for Load Balancing in Fog Computing
Word Count:	1188
Page Count:	9

Signature:	Aryan Kumar
Date:	13th September 2025

Configuration Manual

Aryan Kumar
23310618

1 Introduction

This manual outlines the full process for setting up and running the research project "Hybrid Bacterial Colony Optimization and Particle Swarm Optimization Algorithm for Load Balancing in Fog Computing." It covers all the stages, including environment preparation, library installation, dataset loading, algorithm configuration, and performance result collection. Following these steps, make sure that the experiment can be accurately reproduced using Google Colab Google (2025) or any other Python-based environment.

2 Software and Hardware Requirements

2.1 Software Requirements

- **Google Colab:** This is a cloud-based Jupyter notebook environment that is used for coding, execution, and result visualisation in this project. This Google Colab provides pre-configured Python runtimes and avoids the need for any local installation.
- **Python 3.10:** This is the Programming language used for implementing the BCO, PSO, AIW-PSO, and Hybrid BCO-PSO algorithms in the Google Colab and also used for data processing and visualisation.
- **MEALPY Library (v3.0.1):** This is the Python library for metaheuristic optimisation algorithms Van Thieu and Mirjalili (2023). This library is installed in the Google Colab environment by using `!pip install mealpy==3.0.1`.
- **NumPy (v1.26 or above):** This library is used for the numerical computations and any array operations in the algorithm, with the random number generation in the optimisation process.
- **Matplotlib (v3.8 or above):** This is a library used for generating the convergence curves after the execution of the algorithm.
- **Python Standard Libraries:** `json` (for dataset loading and output file writing), `time` (for execution time measurement) and `random` (for the stochastic components in the algorithm).

2.2 Hardware Specifications

- **Operating System:** The Google Colab virtual machine running on Ubuntu 22.04 LTS. The same code can also run on Windows, macOS, or Linux system.
- **Processor:** Colab's free-tier CPU, which is an Intel Xeon single-core 2.20 GHz. This was suitable for the test scenarios without any performance issues.
- **Memory (RAM):**12 GB available in the Colab environment, which is sufficient for the largest workload tested with 10,000 task count and 10 VMs with 100 bacteria and 50 iterations.
- **Storage:** 68 GB temporary disk space is required in Google Colab for the datasets, intermediate files, and results.
- **GPU/TPU:** Not required; all experiments run entirely on CPU.

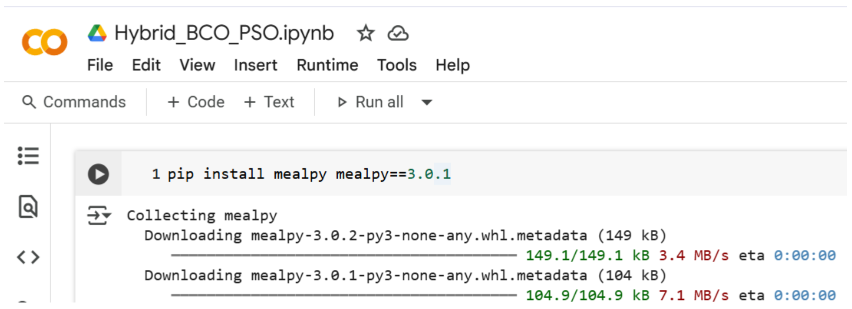
Note: Specifications may vary slightly that is depending on the Colab session but the experiments were successfully reproduced on Colab's free CPU runtime without requiring any GPU or TPU acceleration.

3 Implementation Environment: Google Colab Setup

All the experiments for this project were done using Google Colab, which is a cloud-based Jupyter environment. The optimization algorithms: BCO, PSO, AIW-PSO, and the hybrid BCO-PSO were implemented using Python and the MEALPY library. Synthetic task scheduling datasets were used in JSON format and uploaded manually for each algorithm run.

1. **Open Google Colab:** Go to `https://colab.research.google.com` and start a new notebook by selecting `File > New notebook`.
2. **Install Required Libraries:** Install MEALPY by executing:

   ```
   !pip install mealpy==3.0.1
   ```



```
pip install mealpy mealpy==3.0.1
```

```
Collecting mealpy
  Downloading mealpy-3.0.2-py3-none-any.whl.metadata (149 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.1/149.1 kB 3.4 MB/s eta 0:00:00
  Downloading mealpy-3.0.1-py3-none-any.whl.metadata (104 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.9/104.9 kB 7.1 MB/s eta 0:00:00
```

Figure 1: Installing MEALPY version 3.0.1 in Google Colab

3. **Upload Dataset:** Upload the required JSON dataset files using the upload section in the Google Colab's file explorer (left panel) with the folder icon at the bottom. The uploaded files will appear in the file list as shown in Figure 2.

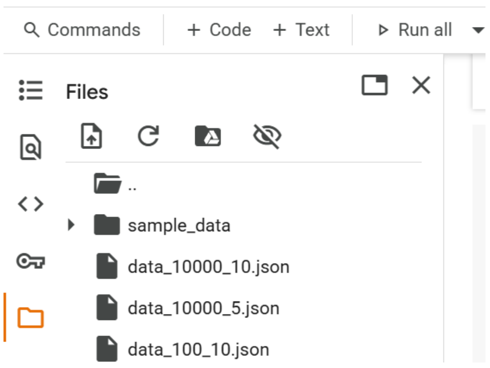
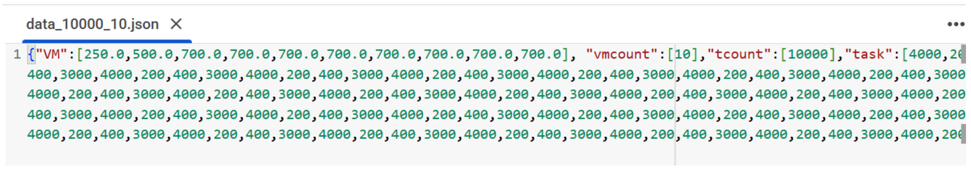


Figure 2: Uploaded dataset files in Google Colab file explorer

4. **Dataset Structure:** The datasets are stored in JSON format that contains:

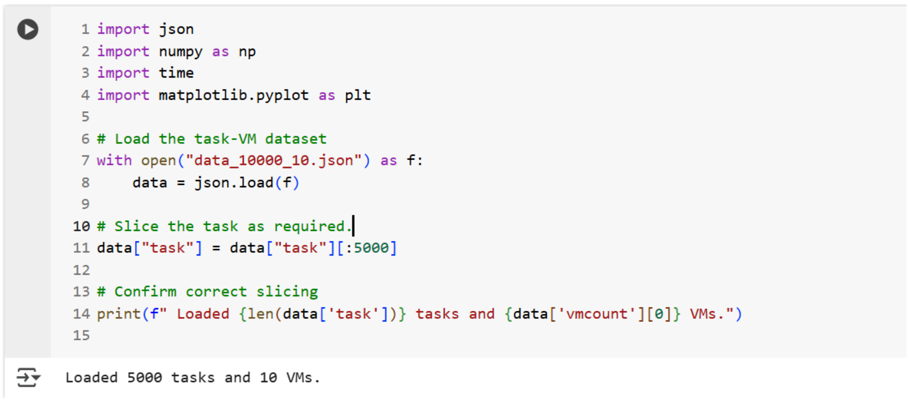
   - **VM**: Virtual machine processing capacities.
   - **vmcount**: Number of VMs.
   - **tcount**: Number of tasks.
   - **task**: List of task lengths (in MI).



```
{"VM":[250.0,500.0,700.0,700.0,700.0,700.0,700.0,700.0,700.0,700.0], "vmcount":[10],"tcount":[10000],"task":[4000,20
400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000
4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200
400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000
4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,200,400,3000,4000,20
```

Figure 3: preview from `data_10000_10.json` showing dataset structure

5. **Load and Preprocess Data:** Parse the selected JSON dataset file, and if needed, the number of tasks can be increased or decreased with a manageable subset. For example, the snippet below loads `data_10000_10.json` and keeps only the first 5000 tasks.



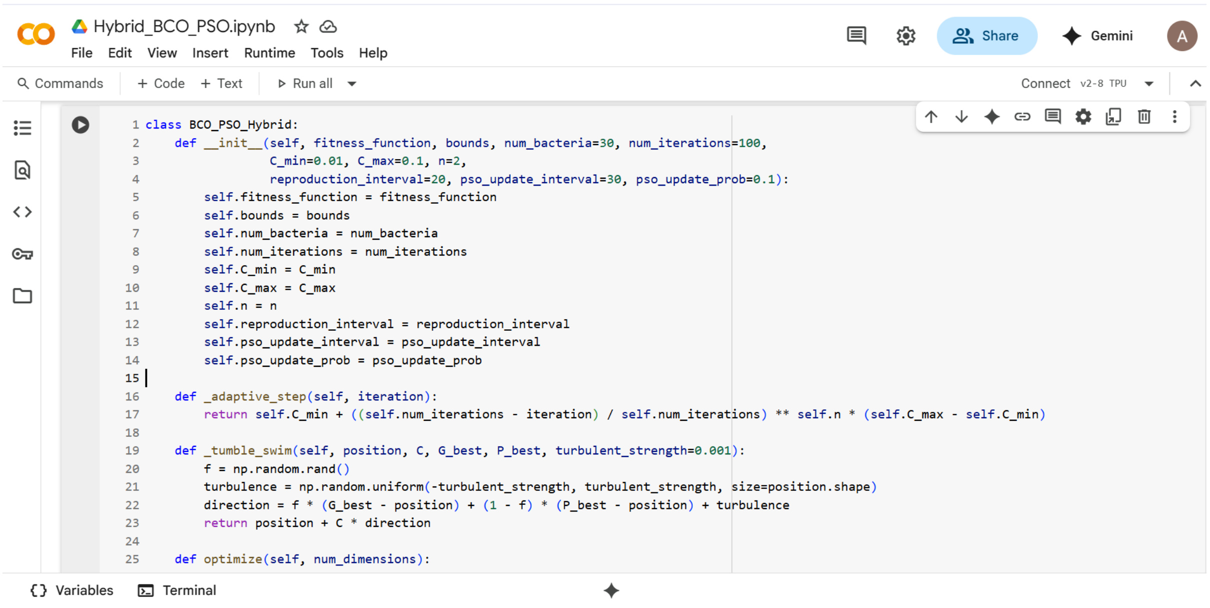
```python
import json
import numpy as np
import time
import matplotlib.pyplot as plt

# Load the task-VM dataset
with open("data_10000_10.json") as f:
    data = json.load(f)

# Slice the task as required.
data["task"] = data["task"][:5000]

# Confirm correct slicing
print(f" Loaded {len(data['task'])} tasks and {data['vmcount'][0]} VMs.")

```

```
Loaded 5000 tasks and 10 VMs.
```

Figure 4: Dataset loaded and sliced, showing confirmation of task and VM counts

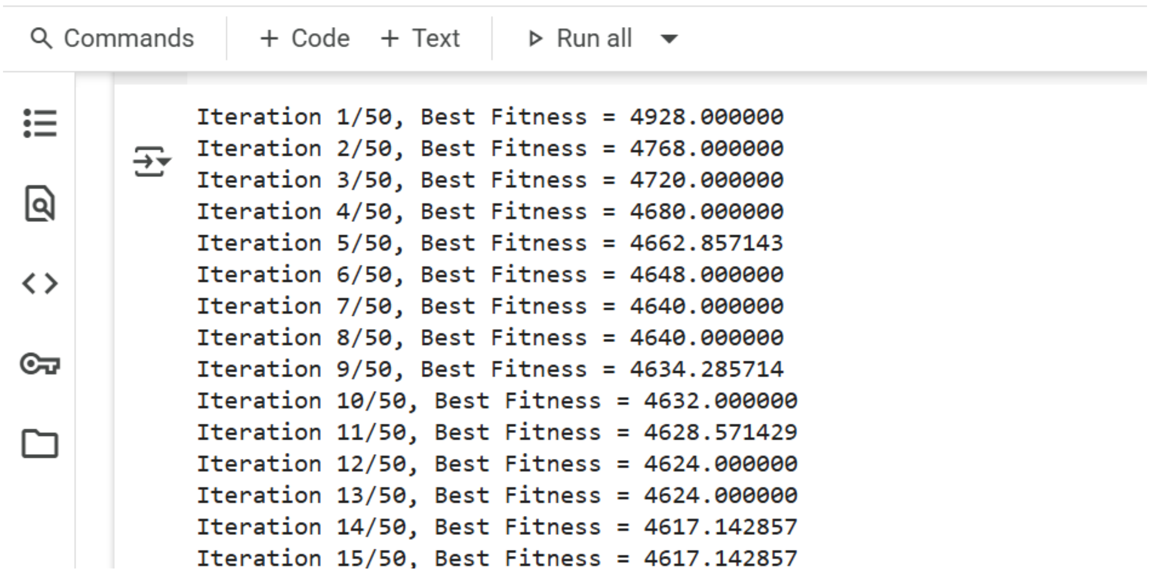
6. **Run Algorithm:**The execution of BCO, PSO, AIW-PSO, and Hybrid BCO-PSO algorithm by running the corresponding cell in the Colab notebook. Figure 6

shows that the optimization process is in progress with the iteration-wise updates of the best fitness (makespan) value. During this execution the console displays the iteration-wise best fitness values available and shows how the solution gets improved over time. Figure 5 shows a preview from the Hybrid BCO-PSO algorithm implementation in the Google Colab environment.



```python
class BCO_PSO_Hybrid:
    def __init__(self, fitness_function, bounds, num_bacteria=30, num_iterations=100,
                 C_min=0.01, C_max=0.1, n=2,
                 reproduction_interval=20, pso_update_interval=30, pso_update_prob=0.1):
        self.fitness_function = fitness_function
        self.bounds = bounds
        self.num_bacteria = num_bacteria
        self.num_iterations = num_iterations
        self.C_min = C_min
        self.C_max = C_max
        self.n = n
        self.reproduction_interval = reproduction_interval
        self.pso_update_interval = pso_update_interval
        self.pso_update_prob = pso_update_prob

    def _adaptive_step(self, iteration):
        return self.C_min + ((self.num_iterations - iteration) / self.num_iterations) ** self.n * (self.C_max - self.C_min)

    def _tumble_swim(self, position, C, G_best, P_best, turbulent_strength=0.001):
        f = np.random.rand()
        turbulence = np.random.uniform(-turbulent_strength, turbulent_strength, size=position.shape)
        direction = f * (G_best - position) + (1 - f) * (P_best - position) + turbulence
        return position + C * direction

    def optimize(self, num_dimensions):
```

Figure 5: Preview from Hybrid BCO–PSO Python implementation



```
Iteration 1/50, Best Fitness = 4928.000000
Iteration 2/50, Best Fitness = 4768.000000
Iteration 3/50, Best Fitness = 4720.000000
Iteration 4/50, Best Fitness = 4680.000000
Iteration 5/50, Best Fitness = 4662.857143
Iteration 6/50, Best Fitness = 4648.000000
Iteration 7/50, Best Fitness = 4640.000000
Iteration 8/50, Best Fitness = 4640.000000
Iteration 9/50, Best Fitness = 4634.285714
Iteration 10/50, Best Fitness = 4632.000000
Iteration 11/50, Best Fitness = 4628.571429
Iteration 12/50, Best Fitness = 4624.000000
Iteration 13/50, Best Fitness = 4624.000000
Iteration 14/50, Best Fitness = 4617.142857
Iteration 15/50, Best Fitness = 4617.142857
```

Figure 6: Iteration-wise best fitness values during execution

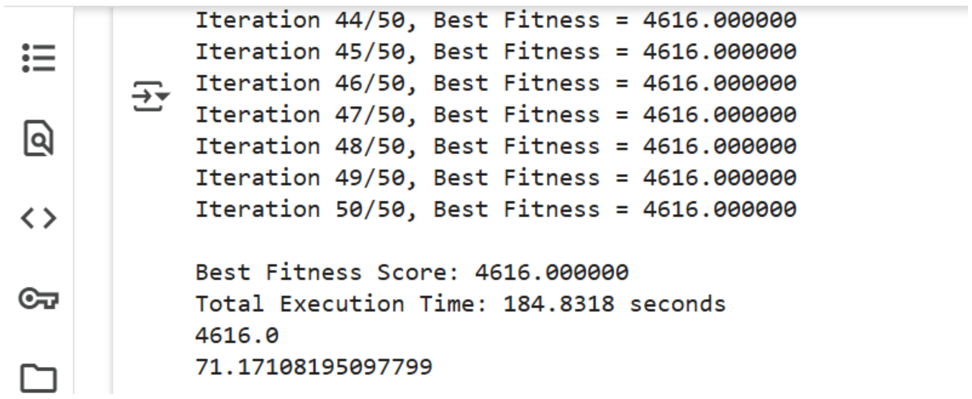
4 Results Collection and Performance Analysis

4.1 Collect Results

After execution of the algorithms the notebook would displays the following key performance metrics:

- **Makespan:** The best fitness score found by the algorithm (in million instructions).
- **Execution Time:** The total runtime of the optimisation process in seconds.
- **Average VM Utilisation:** Percentage of total VM capacity used during scheduling.

Figure 7 shows the final output after all iterations, which includes the best fitness score, execution time, and VM utilisation.



```
Iteration 44/50, Best Fitness = 4616.000000
Iteration 45/50, Best Fitness = 4616.000000
Iteration 46/50, Best Fitness = 4616.000000
Iteration 47/50, Best Fitness = 4616.000000
Iteration 48/50, Best Fitness = 4616.000000
Iteration 49/50, Best Fitness = 4616.000000
Iteration 50/50, Best Fitness = 4616.000000

Best Fitness Score: 4616.000000
Total Execution Time: 184.8318 seconds
4616.0
71.17108195097799
```

Figure 7: Final output showing best fitness score, execution time, and VM utilisation

The notebook also generates a convergence plot (Figure 8) showing how the best fitness value improves over iterations.

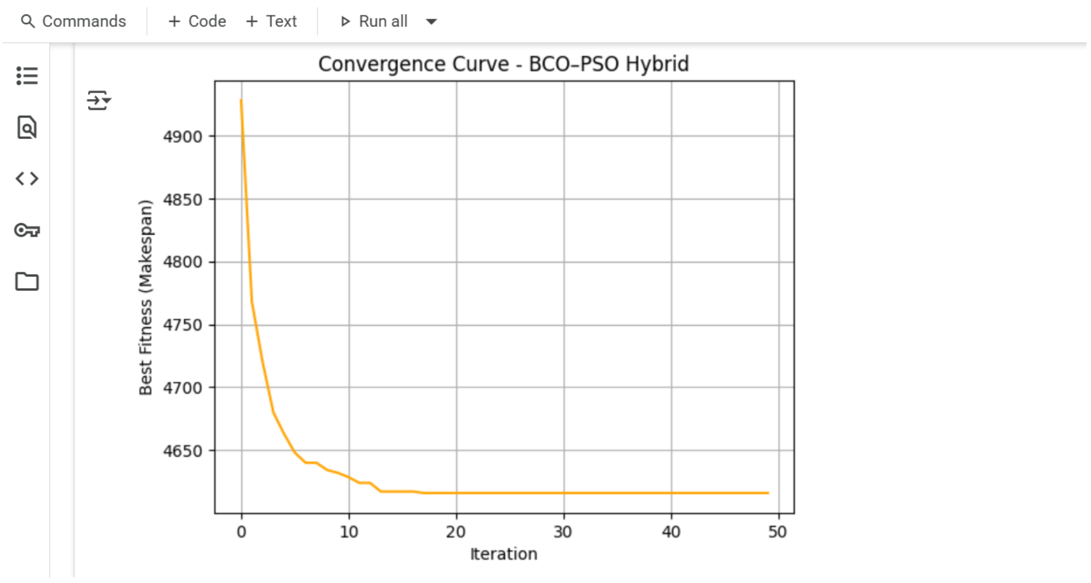
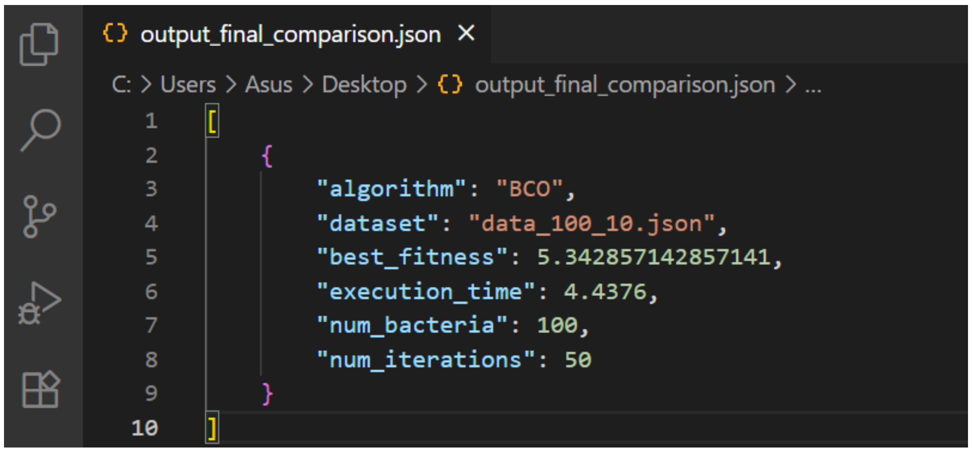


Figure 8: Convergence plot showing best fitness vs. iteration for the Hybrid BCO–PSO algorithm

Additionally, the result of each algorithm run could be seen as in Figure 9 shows an example of the structured JSON output (`output_final_comparison.json`) that is produced after each execution. This file records all the important run parameters and results for each algorithm, which provides a consistent post processing and comparison between the multiple experimental runs.



output_final_comparison.json

C: > Users > Asus > Desktop > output_final_comparison.json > ...

```json
[
    {
        "algorithm": "BCO",
        "dataset": "data_100_10.json",
        "best_fitness": 5.342857142857141,
        "execution_time": 4.4376,
        "num_bacteria": 100,
        "num_iterations": 50
    }
]
```

Figure 9: Sample JSON output file entry for a BCO run with 100 tasks and 10 VMs

This setup helps to makes it easy to repeat the experiments on any computer with internet access without needing to install anything locally. Google Colab provides all the required Python tools and computing power, and the results are saved in a structured JSON file so that they can be compared further and analysed in the Evaluation section.

4.2 Performance Across Task Sizes

The algorithms were tested with 100, 200, 1000, 5000, and 10000 tasks and 10 VMs. Figures 10 to 12 show the changes in makespan, VM utilisation, and execution time across these workloads.

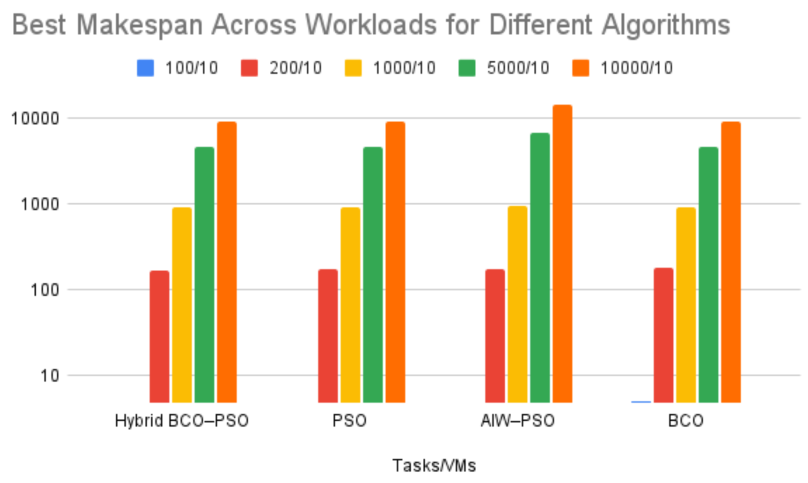


Figure 10: Best makespan for workloads of 100–10000 tasks (10 VMs) across four algorithms. The y-axis uses a logarithmic scale.

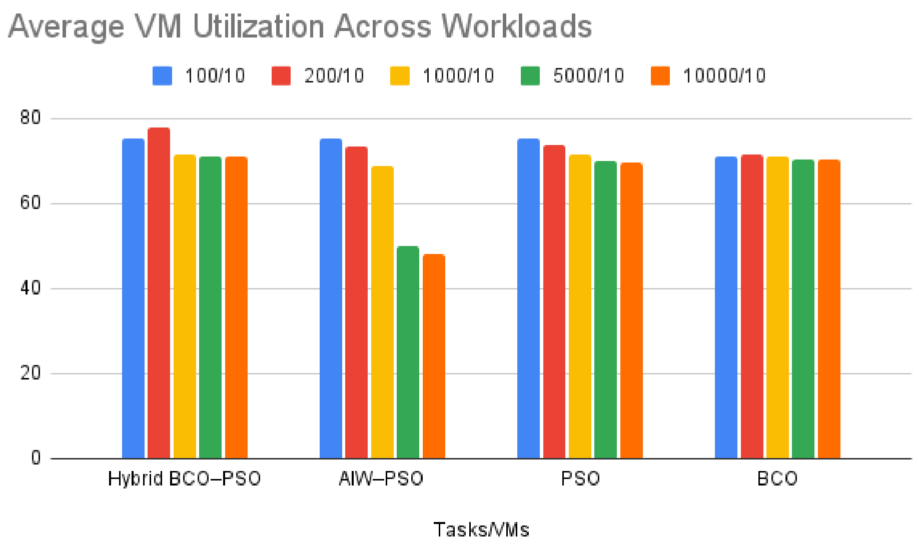


Figure 11: Average VM utilisation for workloads of 100–10000 tasks (10 VMs) across four algorithms.

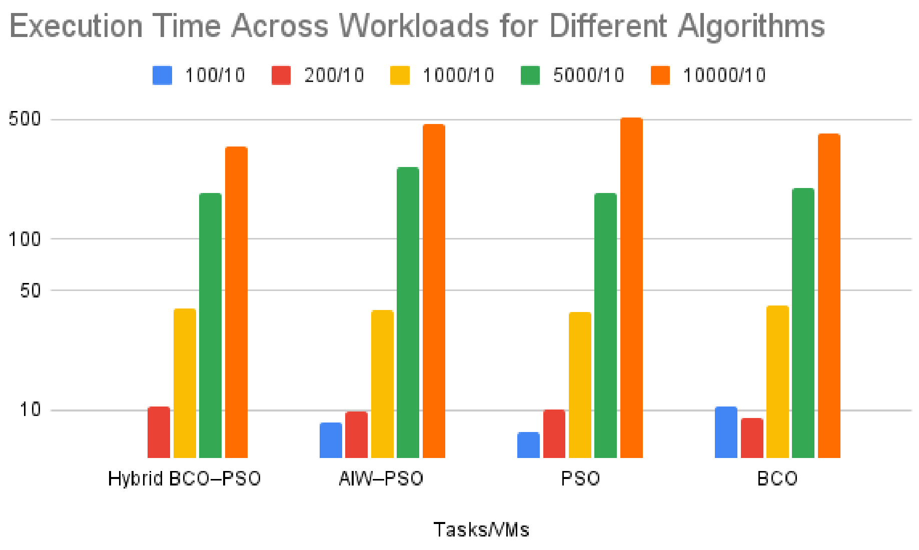


Figure 12: Execution time for workloads of 100–10000 tasks (10 VMs) across four algorithms.

As an example of algorithm behaviour over time Figure 13 shows the convergence curves for a workload in the case of 5000 tasks on 10 VMs. As each subplot represents the tested algorithms: PSO, BCO-PSO Hybrid, BCO, and AIW-PSO, which illustrates how the best fitness value changes over the 50 iterations.

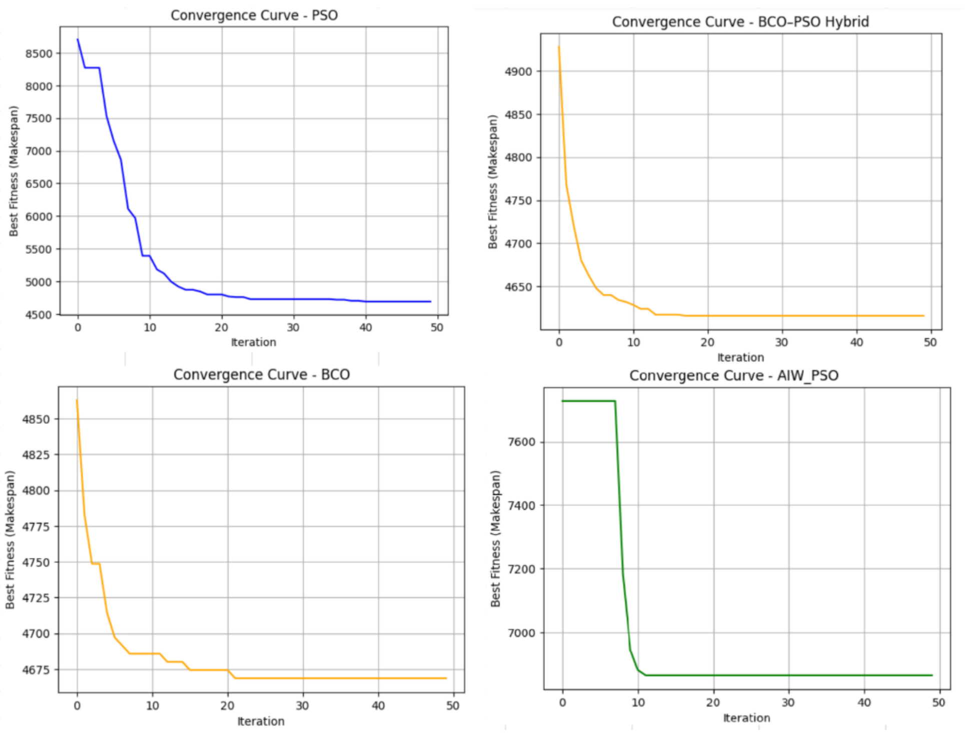


Figure 13: Convergence curves for 5000 tasks and 10 VMs across four algorithms

5 Summary

This configuration manual provides all the necessary steps to set up, execute, and replicate the hybrid BCO-PSO algorithm for load balancing in fog computing environments. It also covers the environment setup in Google Colab, dataset preparation, algorithm execution, and performance result collection. By following the described process, the researcher can reproduce the same results and can compare the algorithm behaviours and can adapt the workflow for further experiments.

Github link for project : https://github.com/aryankumar-nci/FinalProject

6 Troubleshooting

The following are common issues that may occur during execution, and here are some suggested solutions:

- **FileNotFoundError:** For making sure that the required dataset file is uploaded properly into the Colab working directory before running any of those algorithms.
- **ModuleNotFoundError:** If the MEALPY or other required libraries are missing, then re-run the installation cell as:

  ```
  !pip install mealpy==3.0.1 numpy matplotlib
  ```

- **Kernel Reset or Session Timeout:** If the Colab session disconnects or restarts, then re-run all cells from the beginning in a sequential manner.

References

Google (2025). Google colaboratory, `https://colab.research.google.com/`. Accessed: 2025-08-10.

Van Thieu, N. and Mirjalili, S. (2023). MEALPY: An open-source library for latest meta-heuristic algorithms in Python, *Journal of Systems Architecture* **139**: 102871.