National College of Ireland

Configuration Manual

MSc Research Project
MSCCLOUD1$_B$

Zeeshan Ali
Student ID: 23382058

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Zeeshan Ali
Student ID:	23382058
Programme:	MSCCLOUD1$_B$
Year:	2025
Module:	MSc Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	28/01/2026
Project Title:	Configuration Manual
Word Count:	2387
Page Count:	17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Zeeshan
Date:	27th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Zeeshan Ali
23382058

1 System Requirements

This section outlines the hardware, software, and network requirements necessary for deploying and running the configuration management system.

1.1 Hardware Requirements

The following hardware specifications are required for optimal system performance:

- **CPU:** Minimum 4 cores (8 cores recommended for training operations)
- **RAM:** Minimum 8GB (16GB recommended for enhanced performance)
- **Storage:** Minimum 20GB free space for data storage and model persistence
- **GPU:** Optional but recommended for accelerated training operations (CUDA-compatible GPU with 4GB+ VRAM)

1.2 Software Requirements

The system requires the following software components and their respective versions:

- **Operating System:**
  - Linux (Ubuntu 20.04 or higher recommended)
  - macOS 10.15 or higher
  - Windows 10/11 (WSL2 recommended for compatibility)
- **Python:** Version 3.9 or 3.10
- **Kubernetes:** Version 1.20 or higher (required for production deployment)
- **Docker:** Version 20.10 or higher (for containerization)
- **kubectl:** Version 1.20 or higher (for Kubernetes cluster management)

1.3 Network Requirements

The following network configurations and connectivity requirements must be satisfied:

- Internet connection for downloading dependencies and datasets
- Access to Kubernetes cluster (local or remote deployment)
- Port 8000 available for LSTM API service
- Port 30080 available for web application (when using NodePort service type)

2 Installation Guide

This section provides step-by-step instructions for setting up the AI load balancer system, including repository cloning, environment configuration, and dependency installation.

2.1 Clone the Repository

Begin by cloning the project repository and navigating to the project directory:

```
git clone <repository-url>
cd ai-load-balancer
```

2.2 Create Virtual Environment

Create and activate a Python virtual environment to isolate project dependencies.

Linux/macOS:

```
python3 -m venv venv
source venv/bin/activate
```

Windows:

```
python -m venv venv
venv\Scripts\activate
```

2.3 Install Python Dependencies

Upgrade pip and install all required Python packages from the requirements file:

```
pip install --upgrade pip
pip install -r requirements.txt
```

2.4 Verify Installation

Confirm that Python and all required packages are correctly installed:

```
python --version # Should show Python 3.9 or 3.10
pip list # Verify all packages are installed
```

2.5 Download Dataset (Optional)

If using Google Cluster Trace data for training or testing, follow these steps:

```
# Follow instructions in DATASET_DOWNLOAD.md
# Place data files in data/raw/ directory
python process_google_cluster_data.py
```

Note: Refer to the `DATASET_DOWNLOAD.md` file for detailed instructions on obtaining and preparing the dataset.

3 Environment Configuration

This section details the configuration of environment variables, directory structure, and Kubernetes cluster setup required for the system deployment.

3.1 Environment Variables

Create a `.env` file in the project root directory to configure system parameters. If not created, default values will be used:

```
# LSTM API Configuration
PORT=8000
MODEL_PATH=models/lstm_best_model.h5

# Kubernetes Configuration
KUBERNETES_NAMESPACE=default
USE_SIMULATOR=false

# DDPG Configuration
TENSORBOARD_LOG_DIR=logs/ddpg
OUTPUT_DIR=experiments
MODEL_SAVE_DIR=models

# Data Paths
DATA_PATH=data/cluster_workload.csv
PREPROCESSED_DATA_PATH=data/preprocessed_data.npz
LSTM_MODEL_PATH=models/lstm_best_model.h5
```

3.2 Directory Structure

The following directories are created automatically if they do not exist:

- `models/` – Trained model files
- `experiments/` – Experiment results and logs
- `logs/ddpg/` – TensorBoard logs for DDPG training
- `data/` – Dataset files
- `data/raw/` – Raw dataset files
- `data/preprocessed/` – Preprocessed data files

3.3 Kubernetes Configuration

Configure Kubernetes cluster access depending on your deployment environment.

3.3.1 Local Development (Minikube)

For local development and testing using Minikube:

```
minikube start
kubectl config use-context minikube
```

3.3.2 Remote Cluster

For production deployment on a remote Kubernetes cluster:

```
# Configure kubectl with cluster credentials
kubectl config set-cluster <cluster-name>
kubectl config set-credentials <user>
kubectl config set-context <context-name>
kubectl config use-context <context-name>
```

3.3.3 Verify Kubernetes Connection

Confirm that kubectl is properly configured and can communicate with the cluster:

```
kubectl cluster-info
kubectl get nodes
```

4 Dependencies Setup

This section outlines the installation and configuration of all required dependencies, including Python packages, system-level dependencies, and Kubernetes client libraries.

4.1 Python Package Dependencies

All Python dependencies are specified in the `requirements.txt` file:

```
tensorflow==2.15.0
pandas
numpy
matplotlib
seaborn
scikit-learn
jupyter
notebook
flask
fastapi
uvicorn
prometheus-client
requests
kubernetes
```

4.2 System Dependencies

Install system-level dependencies based on your operating system.

Linux (Ubuntu/Debian):

```
sudo apt-get update
sudo apt-get install -y gcc g++ python3-dev
```

macOS:

```
# Install Xcode Command Line Tools
xcode-select --install
```

Windows:

- Install Visual Studio Build Tools
- Install Microsoft C++ Build Tools

4.3 Kubernetes Python Client

The Kubernetes Python client is automatically installed via `requirements.txt`. Ensure you have one of the following configurations:

- Valid `~/.kube/config` file (for local development)
- Running inside Kubernetes cluster (for in-cluster configuration)

5 Kubernetes Configuration

This section provides detailed instructions for deploying and managing Kubernetes resources, including the web application, LSTM prediction service, and namespace configuration.

5.1 Deployment Files

Deployment YAML files are located in the `k8s/` directory:

- `web-app-deployment.yaml` – Web application deployment
- `lstm-deployment.yaml` – LSTM prediction service
- `test-app-deployment.yaml` – Test application

5.2 Deploy Web Application

Deploy the web application using the following command:

```
kubectl apply -f k8s/web-app-deployment.yaml
```

This creates:

- Deployment: `web-app` with 3 replicas
- Service: `web-app-service` (NodePort on port 30080)

5.3 Deploy LSTM Prediction Service

5.3.1 Build Docker Image

Build the Docker image for the LSTM prediction service:

```
docker build -t lstm-prediction-api:latest .
```

For Minikube:

```
eval $(minikube docker-env)
docker build -t lstm-prediction-api:latest .
```

5.3.2 Deploy to Kubernetes

Deploy the LSTM service to the Kubernetes cluster:

```
kubectl apply -f k8s/lstm-deployment.yaml
```

5.4 Verify Deployments

Verify that all resources are properly deployed and running:

```
# Check deployments
kubectl get deployments

# Check pods
kubectl get pods

# Check services
kubectl get services

# View logs
kubectl logs -f deployment/lstm-prediction-service
```

5.5 Namespace Configuration

The default namespace is `default`. To use a custom namespace, follow these steps:

Step 1: Create namespace

```
kubectl create namespace <namespace-name>
```

Step 2: Update deployment files to include namespace, or use:

```
kubectl apply -f k8s/web-app-deployment.yaml -n <namespace-name>
```

Step 3: Update configuration in `ddpg_config.py`

```
KUBERNETES_NAMESPACE = '<namespace-name>'
```

6 Model Configuration

This section describes the configuration parameters for the DDPG agent and LSTM model, including network architecture, training parameters, and reward function settings.

6.1 DDPG Agent Configuration

All DDPG hyperparameters are configured in `ddpg_config.py`.

6.1.1 Network Architecture

```
ACTOR_HIDDEN_LAYERS = [256, 256] # Actor network layers
CRITIC_HIDDEN_LAYERS = [256, 256] # Critic network layers
ACTOR_LEARNING_RATE = 1e-4
CRITIC_LEARNING_RATE = 1e-3
```

6.1.2 Training Parameters

```
BATCH_SIZE = 64
BUFFER_SIZE = 100000 # Experience replay buffer
GAMMA = 0.99 # Discount factor
TAU = 0.005 # Soft update coefficient
NOISE_SCALE = 0.1 # Initial exploration noise
NOISE_DECAY = 0.995 # Noise decay per episode
MIN_NOISE = 0.01 # Minimum noise
```

6.1.3 Training Schedule

```
MAX_EPISODES = 100
MAX_STEPS_PER_EPISODE = 200
WARMUP_STEPS = 200 # Random actions before training
UPDATE_FREQUENCY = 1 # Network update frequency
```

6.1.4 State Space

```
STATE_DIM = 6 # [cpu, memory, pred_cpu, pred_mem, pred_rate, num_pods]
STATE_FEATURES = [
    'current_cpu_usage',
    'current_memory_usage',
    'predicted_cpu_usage',
    'predicted_memory_usage',
    'predicted_request_rate',
    'current_num_pods'
]
```

6.1.5 Action Space

```
ACTION_DIM = 1 # Continuous scaling factor
ACTION_MIN = -2.0 # Maximum scale down (remove 2 pods)
ACTION_MAX = 5.0 # Maximum scale up (add 5 pods)
```

6.1.6 Reward Function

```
TARGET_LATENCY_MS = 100.0 # Target latency in milliseconds
MAX_LATENCY_MS = 1000.0 # Maximum acceptable latency
LATENCY_WEIGHT = 1.0    # Weight for latency penalty
ENERGY_WEIGHT = 0.1     # Weight for energy cost
SLA_VIOLATION_WEIGHT = 10.0 # Weight for SLA violations
SCALING_PENALTY = 0.01  # Penalty for frequent scaling
```

6.1.7 Pod Constraints

```
MIN_PODS = 1
MAX_PODS = 20
INITIAL_PODS = 3
POD_CAPACITY_RPS = 100.0 # Requests per pod per second
BASE_LATENCY_MS = 50.0 # Base latency with no load
```

6.2 LSTM Model Configuration

LSTM model configuration is embedded in the training script (`week3_lstm_training.py`):

- Sequence Length: 60 timesteps
- Features: 3 (CPU usage, Memory usage, Request rate)
- Model Architecture: LSTM layers with dropout
- Model Path: `models/lstm_best_model.h5`

6.3 Model Paths

```
# Data paths
DATA_PATH = 'data/cluster_workload.csv'
PREPROCESSED_DATA_PATH = 'data/preprocessed_data.npz'
LSTM_MODEL_PATH = 'models/lstm_best_model.h5'

# Output paths
OUTPUT_DIR = 'experiments'
MODEL_SAVE_DIR = 'models'
TENSORBOARD_LOG_DIR = 'logs/ddpg'
```

7 API Configuration

This section details the configuration and usage of the LSTM prediction API, including endpoints, request formats, and URL configuration.

7.1 LSTM Prediction API

The LSTM API is a FastAPI service running on port 8000.

Endpoints:

- `GET /` – API information
- `GET /health` – Health check
- `POST /predict` – Make predictions
- `GET /model/info` – Model information
- `GET /docs` – Interactive API documentation

Configuration:

- Port: 8000 (configurable via `PORT` environment variable)
- Model Path: `models/lstm_best_model.h5` (configurable via `MODEL_PATH`)
- Sequence Length: 60 timesteps

Start API:

```
python lstm_api.py
# Or with custom port:
PORT=8001 python lstm_api.py
```

Using Docker:

```
docker run -p 8000:8000 -v $(pwd)/models:/app/models \
    lstm-prediction-api:latest
```

7.2 API Request Format

Prediction Request:

```
{
  "sequence": [
    [0.3, 0.25, 0.3],
    [0.35, 0.28, 0.35],
    // ... 60 timesteps total
    [0.4, 0.3, 0.4]
  ]
}
```

Prediction Response:

```
{
  "predictions": [0.42, 0.32, 0.42],
  "confidence": 0.85
}
```

7.3 LSTM API URL Configuration

In `ddpg_config.py`:

```
LSTM_API_URL = 'http://localhost:8000' # Local development
# For Kubernetes: 'http://lstm-prediction-service:8000'
```

In `orchestrator.py`:

```
orchestrator = AutoscalingOrchestrator(
    deployment_name='web-app',
    lstm_api_url='http://lstm-prediction-service:8000', # Kubernetes service
        name
    use_simulator=False
)
```

8 Running Instructions

This section provides step-by-step instructions for training models, running the orchestration service, generating load, and evaluating system performance.

8.1 Training the LSTM Model

```
python week3_lstm_training.py
```

This will:

- Load and preprocess data
- Train LSTM model
- Save model to `models/lstm_best_model.h5`
- Generate training plots in `experiments/`

8.2 Training the DDPG Agent

```
python week5_ddpg_training.py
```

This will:

- Load trained LSTM model
- Initialize DDPG agent
- Train agent using Kubernetes simulator
- Save model weights to `models/`
- Generate training metrics in `experiments/`

Monitor Training:

```
tensorboard --logdir=logs/ddpg
# Open http://localhost:6006 in browser
```

8.3 Running the Orchestration Service

Start LSTM API:

```
python lstm_api.py
```

In another terminal, start orchestrator:

```
python orchestrator.py
```

Or use the orchestrator programmatically:

```
from orchestrator import AutoscalingOrchestrator

orchestrator = AutoscalingOrchestrator(
    deployment_name='web-app',
    lstm_api_url='http://localhost:8000',
    use_simulator=False
)

# Run for 10 minutes with 30-second intervals
orchestrator.run_continuous(interval=30, duration=600)
```

8.4 Running with Simulator

For testing without Kubernetes:

```
orchestrator = AutoscalingOrchestrator(
    deployment_name='web-app',
    lstm_api_url='http://localhost:8000',
    use_simulator=True # Use simulator
)
```

8.5 Running Load Generator

```
from load_generator import LoadGenerator

# Initialize load generator
generator = LoadGenerator(
    target_url='http://localhost:30080', # Web app URL
    pattern='variable'
)

# Start load generation
generator.start(
    pattern='variable',
    base_rps=10,
    max_rps=50,
    duration=300
)

# Get statistics
```

```
stats = generator.get_stats()
print(f"Average latency: {stats['avg_latency_ms']}ms")
print(f"P95 latency: {stats['p95_latency_ms']}ms")

# Stop generator
generator.stop()
```

8.6 Running Evaluation

```
python week8_evaluation.py
```

This evaluates the system against baseline methods and generates comparison reports.

9 Configuration Parameters

This section provides a comprehensive reference of all configuration parameters used throughout the system.

9.1 DDPG Configuration Summary

9.2 LSTM Configuration

9.3 Kubernetes Configuration

9.4 API Configuration

10 Troubleshooting

This section provides solutions to common issues encountered during installation, configuration, and operation of the system.

10.1 Common Issues

10.1.1 Issue: Kubernetes Connection Failed

Symptoms:

```
Error: Failed to load Kubernetes config
```

Solutions:

1. Verify kubectl configuration:

```
kubectl cluster-info
kubectl get nodes
```

2. For local development, ensure Minikube is running:

```
minikube status
minikube start
```

Parameter	Default Value	Description
ACTOR_HIDDEN_LAYERS	[256, 256]	Actor network architecture
CRITIC_HIDDEN_LAYERS	[256, 256]	Critic network architecture
ACTOR_LEARNING_RATE	1e-4	Actor learning rate
CRITIC_LEARNING_RATE	1e-3	Critic learning rate
BATCH_SIZE	64	Training batch size
BUFFER_SIZE	100000	Replay buffer size
GAMMA	0.99	Discount factor
TAU	0.005	Soft update coefficient
NOISE_SCALE	0.1	Initial exploration noise
NOISE_DECAY	0.995	Noise decay rate
MIN_NOISE	0.01	Minimum noise level
MAX_EPISODES	100	Maximum training episodes
MAX_STEPS_PER_EPISODE	200	Steps per episode
WARMUP_STEPS	200	Random action steps
ACTION_MIN	-2.0	Minimum scaling action
ACTION_MAX	5.0	Maximum scaling action
TARGET_LATENCY_MS	100.0	Target latency (ms)
MAX_LATENCY_MS	1000.0	Maximum latency (ms)
LATENCY_WEIGHT	1.0	Latency penalty weight
ENERGY_WEIGHT	0.1	Energy cost weight
SLA_VIOLATION_WEIGHT	10.0	SLA violation weight
MIN_PODS	1	Minimum pod count
MAX_PODS	20	Maximum pod count
INITIAL_PODS	3	Initial pod count
POD_CAPACITY_RPS	100.0	Pod capacity (req/sec)

Table 1: DDPG Configuration Parameters

3. Check kubeconfig file:

```
cat ~/.kube/config
```

10.1.2 Issue: LSTM Model Not Found

Symptoms:

```
FileNotFoundError: models/lstm_best_model.h5
```

Solutions:

1. Train the LSTM model first:

```
python week3_lstm_training.py
```

2. Verify model file exists:

```
ls -lh models/lstm_best_model.h5
```

3. Check `MODEL_PATH` environment variable or config

10.1.3 Issue: DDPG Model Weights Not Found

Symptoms:

Parameter	Value	Description
SEQ_LENGTH	60	Input sequence length
FEATURES	3	Number of features (CPU, Memory, Request Rate)
MODEL_PATH	models/lstm_best_model.h5	Model file path

Table 2: LSTM Configuration Parameters

Parameter	Default	Description
NAMESPACE	default	Kubernetes namespace
DEPLOYMENT_NAME	web-app	Target deployment name
LSTM_SERVICE_URL	http://lstm-prediction-service:8000	LSTM API service URL

Table 3: Kubernetes Configuration Parameters

```
Warning: No trained model found. Using untrained agent.
```

Solutions:

1. Train the DDPG agent:

```
python week5_ddpg_training.py
```

2. Verify model files exist:

```
ls -lh models/ddpg_agent_*.h5
```

10.1.4 Issue: LSTM API Connection Failed

Symptoms:

```
ConnectionError: Connection refused
```

Solutions:

1. Verify LSTM API is running:

```
curl http://localhost:8000/health
```

2. Check if port 8000 is available:

```
lsof -i :8000
```

3. For Kubernetes, verify service:

```
kubectl get svc lstm-prediction-service
kubectl get pods -l app=lstm-prediction
```

10.1.5 Issue: Metrics API Not Available

Symptoms:

```
Warning: Metrics API not available, using fallback
```

Solutions:

1. Install metrics-server in Kubernetes:

Parameter	Default	Description
PORT	8000	LSTM API port
MODEL_PATH	models/lstm_best_model.h5	LSTM model path
HOST	0.0.0.0	API host address

Table 4: API Configuration Parameters

```
kubectl apply -f https://github.com/kubernetes-sigs/\
metrics-server/releases/latest/download/components.yaml
```

2. For Minikube:

```
minikube addons enable metrics-server
```

3. Verify metrics-server is running:

```
kubectl get deployment metrics-server -n kube-system
```

10.1.6 Issue: Out of Memory During Training

Symptoms:

```
ResourceExhaustedError: OOM when allocating tensor
```

Solutions:

1. Reduce batch size in `ddpg_config.py`:

```
BATCH_SIZE = 32 # Reduce from 64
```

2. Reduce buffer size:

```
BUFFER_SIZE = 50000 # Reduce from 100000
```

3. Reduce network size:

```
ACTOR_HIDDEN_LAYERS = [128, 128] # Reduce from [256, 256]
CRITIC_HIDDEN_LAYERS = [128, 128]
```

10.1.7 Issue: Pod Scaling Not Working

Symptoms:

```
Error scaling deployment: 403 Forbidden
```

Solutions:

1. Verify RBAC permissions:

```
kubectl auth can-i update deployments --namespace=default
```

2. Create ServiceAccount with proper permissions:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: autoscaler
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: autoscaler-role
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "update", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: autoscaler-binding
subjects:
- kind: ServiceAccount
  name: autoscaler
roleRef:
  kind: Role
  name: autoscaler-role
  apiGroup: rbac.authorization.k8s.io
```

10.2 Debugging Tips

1. **Enable Verbose Logging:**

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

2. **Check Pod Logs:**

```
kubectl logs -f deployment/web-app
kubectl logs -f deployment/lstm-prediction-service
```

3. **Monitor Resource Usage:**

```
kubectl top nodes
kubectl top pods
```

4. **Test API Endpoints:**

```
# LSTM API health check
curl http://localhost:8000/health

# Test prediction
curl -X POST http://localhost:8000/predict \
  -H "Content-Type: application/json" \
  -d '{"sequence": [[0.3,0.25,0.3]]*60}'
```

5. **Use Simulator Mode:** For testing without Kubernetes, set `use_simulator=True` in orchestrator initialization.

10.3 Performance Tuning

1. **Increase Training Episodes:**

```
MAX_EPISODES = 200 # Increase from 100
```

2. **Adjust Learning Rates:**

```
ACTOR_LEARNING_RATE = 5e-5 # Lower for stability
CRITIC_LEARNING_RATE = 5e-4
```

3. **Tune Reward Weights:**

```
LATENCY_WEIGHT = 2.0 # Increase for stricter latency control
ENERGY_WEIGHT = 0.05 # Decrease to prioritize performance
```

4. **Adjust Pod Limits:**

```
MIN_PODS = 2 # Increase minimum for better availability
MAX_PODS = 30 # Increase maximum for higher capacity
```