National College of Ireland

Configuration Manual

MSc Research Project
Master of Science in Cloud Computing

Pavan Binu
Student ID: 23199814

School of Computing
National College of Ireland

Supervisor: Ahmed Makki

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pavan Binu
Student ID:	23199814
Programme:	Master of Science in Cloud Computing **Year:** 2024-2025
Module:	MSc Research Project
Lecturer:	Ahmed Makki
Submission Due Date:	15-09-2025
Project Title:	Enhancing Cloud Service Efficiency with Predictive AutoScaling: A Machine Learning-Based Approach

Word Count:1565 Page Count:11

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: Pavan Binu

Date: 15-09-2025

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

Pavan Binu
23199814

1 Introduction

The primary objective of this document is to provide an overview of all the steps included in the Research Project. This document covers the detailed instructions for each module which are necessary to understand the working of the project. This involves a detailed explanation on Data Preprocessing, Model training, Hybrid Integration, evaluation and analysis of the models. The steps explained in this report will help the reader to understand the working of this project.

2 Prerequisites

A basic knowledge on python programming, including working with libraries, functions and data structures is required. Familiarity with Machine Learning algorithms, and concepts such as LSTM (Long Short-Term Memory), CNN (Convolutional Neural Networks), DQN (Deep Q-Networks) will be beneficial, still not mandatory.

3 Data Processing and Analysis Framework

3.1 Overview of Data Processing Pipeline

A complete framework for data analysis of massive datasets extracted from multiple compressed archive files is presented in this paper. The paper describes how to extract and process and analyze seven different.gz compressed files which contain various tables that describe machine learning and system performance data.

The process starts with automated data extraction from compressed archives before proceeding to the preprocessing stage which checks data consistency for different analytical models. The data sets undergo strict validation procedures to ensure consistency and reliability from start to finish of the analysis pipeline (Beloglazov and Buyya 2015). The preprocessing stage contains steps to deal with missing values and normalize data distributions and prepare feature sets that are optimal for machine learning applications.

3.1.1 Loading Data from Compressed Files

```python
import gzip
import pandas as pd
import os

# Define the list of uploaded .gz file paths
uploaded_files = [
    '/content/sample_data/pai_group_tag_table.tar.gz',
    '/content/sample_data/pai_instance_table.tar.gz',
    '/content/sample_data/pai_job_table.tar.gz',
    '/content/sample_data/pai_machine_metric.tar.gz',
    '/content/sample_data/pai_machine_spec.tar.gz',
    '/content/sample_data/pai_sensor_table.tar.gz',
    '/content/sample_data/pai_task_table.tar.gz'
]

# Create a dictionary to store the loaded DataFrames
extracted_csv_paths = {}

# Iterate through the list of file paths and load the data into DataFrames
for file_path in uploaded_files:
    try:
        base_name = os.path.basename(file_path).replace('.csv.gz', '')
        with gzip.open(file_path, 'rt') as f:
            df = pd.read_csv(f)
        extracted_csv_paths[base_name] = df
        print(f"Successfully loaded {file_path}")
    except FileNotFoundError:
        print(f"Error: File not found at {file_path}")
    except Exception as e:
        print(f"An error occurred while processing {file_path}: {e}")
```

(Figure 1: Loading Data)

3.2 Data Loading and Extraction Procedures

The Python gzip library performs CSV file extraction from compressed archives during data loading operations. The solution includes error handling features that maintain data provenance during extraction operations. Each file receives individual processing steps which maintain the file origin and extraction timestamp and data validation checksum information.

The data extraction procedure follows established guidelines for large datasets using memory-efficient methods to optimize computational resource utilization. The data processing system includes error logging and exception handling that maintains robust processing of corrupted or incomplete archive files (Feng and Buyya 2016).

The main aspects to consider when loading data include:

- The system verifies each compressed file for integrity before extraction to prevent data loss and corruption during processing steps.

- The system uses streaming data processing to handle large datasets without overloading system memory during concurrent processing of multiple gigabyte-scale compressed archives.
- The analytical results become fully reproducible through logging mechanisms that track dataset component source and extraction time and processing status.

```
# Display basic information about the loaded datasets
print("Dataset Information:")
for df_name, df in extracted_csv_paths.items():
    print(f"\n{df_name}:")
    print(f" Shape: {df.shape}")
    print(f" Columns: {list(df.columns)}")
    print(f" Data types: {df.dtypes.value_counts()}")
    print(f" Missing values: {df.isnull().sum().sum()}")
```

(*Figure 2: Basic Information from the loaded dataset)*

3.3 Feature Engineering and Preprocessing

The preprocessing pipeline implements dataset-specific feature engineering methods for each data type. The machine learning applications require categorical variable encoding and numerical feature standardization to achieve optimal model results. The preprocessing framework accepts multiple data types which include temporal sequences and categorical identifiers and continuous numerical measurements (Gomes *et al.* 2015).

The statistical validation procedures help detect and resolve outliers and missing values together with data inconsistencies. The framework adjusts its preprocessing methods based on different analytical models to optimize the quality of input data for CNN, LSTM, and DQN model architectures.

4 Machine Learning Model Architecture and Implementation

4.1 Convolutional Neural Network (CNN) Configuration

The CNN implementation employs a hierarchical structure that identifies spatial patterns within large input data dimensions. Multiple convolutional layers with increasing feature map

dimensions are implemented in this model followed by max-pooling operations to reduce dimensions and consolidate features.

The architectural design prioritizes efficient parameters alongside enough complexity to detect complex data patterns. The strategic placement of dropout regularization layers serves to stop overfitting because it is essential for working with small training datasets (Kune et al., 2016). The CNN configuration includes:

4.1.1 Layer Architecture Details:

- The first convolutional layer starts with 32 filters of size 3 which use ReLU activation functions for non-linear transformations. The configuration balances feature extraction capability with low computational requirements.
- The network contains pooling layers that decrease spatial dimensions by 50% using a pool size of 2 while preserving essential information patterns.
- The dense layers convert the extracted features into class probability distributions using softmax activation to perform multi-class classification.

4.1.2 CNN Model for Sequential Data

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout

# Define the CNN model architecture
cnn_model = Sequential([
    Conv1D(32, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Conv1D(64, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),
    Dense(100, activation='relu'),
    Dropout(0.5),
    Dense(len(y_train.unique()), activation='softmax')
])

# Compile the model
cnn_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

(Figure 3: CNN)

4.2 Long Short-Term Memory (LSTM) Network Design

The LSTM architecture provides a solution for sequential data analysis by implementing memory components to effectively handle temporal dependencies in the dataset. The network contains a single LSTM layer with 100 hidden units that preserves long-range dependencies while keeping the computational requirements reasonable.

The model implements multiple dropout mechanisms at different levels to enhance generalization ability and prevent training sequence overfitting. The LSTM configuration achieves a balance between model complexity and training efficiency to provide robust performance in dealing with sequences of different lengths and patterns (Beloglazov and Buyya, 2015).

4.2.1 LSTM Configuration Parameters:

- The 100-unit LSTM layer in this configuration provides enough capacity to detect complex temporal relationships without creating excessive training or inference overhead.
- The LSTM output receives a dropout rate of 0.5 which deactivates 50% of neurons randomly during training iterations to prevent overfitting.
- The last layer applies softmax activation for multi-class probability distribution generation while the previous layer contains 50 and 4 units dense layers which transform LSTM outputs into final classification results.

4.2.2 LSTM Model Configuration

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Reshape

# Define the LSTM model
lstm_model = Sequential([
    # Reshape layer to add time dimension
    Reshape((1, X_train.shape[1]), input_shape=(X_train.shape[1],)),
    LSTM(100, return_sequences=False),
    Dropout(0.5),
    Dense(50, activation='relu'),
    Dense(len(y_train.unique()), activation='softmax')
])

# Compile the LSTM model
lstm_model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

(Figure 4: LSTM)

4.3 Deep Q-Network (DQN) Implementation Framework

The DQN approach unites supervised learning with reinforcement learning principles to develop a hybrid system that implements Q-learning algorithms for enhanced decision making. The architecture uses experience replay and epsilon-greedy exploration to optimize learning efficiency.

The DQN framework generates simulated environmental interactions through synthetic episode production to enable the model to learn optimal action-value mappings without needing environment specifications. This approach proves that reinforcement learning techniques can be used successfully in traditional supervised learning scenarios (Feng and Buyya, 2016).

The architecture of DQN consists of the following components:

- The input states from the preprocessed data are represented as feature vectors which have the same dimensionality as the original data feature space.
- The action space defines the classification categories found in the target variable which allows direct Q-value mapping to class predictions.
- The experience replay buffer functions as a circular buffer which maintains 10,000 entries to store state-action-reward transitions for batch learning that improves training sample efficiency.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

# Define the DQN-like model (using Dense layers for classification)
dqn_model = Sequential([
    Flatten(input_shape=(X_train.shape[1], 1)),
    # Adding Dense layers similar to a Q-network
    Dense(128, activation='relu'),
    Dense(128, activation='relu'),
    Dense(len(y_train.unique()), activation='softmax')
])

# Compile the model
dqn_model.compile(optimizer=Adam(learning_rate=0.001), # Using Adam optimizer with a learning rate
                  loss='sparse_categorical_crossentropy', # Use sparse_categorical_crossentropy for integer labels
                  metrics=['accuracy'])
```

(Figure 5: DQN)

4.4 Hybrid Model Architecture

```python
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Concatenate, Reshape, LSTM, Dense, Conv1D, MaxPooling1D, Flatten

# Define input layer
input_layer = Input(shape=(X_train.shape[1],), name='hybrid_input')

# CNN branch
cnn_reshape = Reshape((X_train.shape[1], 1), name='cnn_reshape')(input_layer)
cnn_conv1 = Conv1D(32, 3, activation='relu', name='hybrid_cnn_conv1d_4')(cnn_reshape)
cnn_pool1 = MaxPooling1D(2, name='hybrid_cnn_maxpooling1d_4')(cnn_conv1)
cnn_conv2 = Conv1D(64, 3, activation='relu', name='hybrid_cnn_conv1d_5')(cnn_pool1)
cnn_pool2 = MaxPooling1D(2, name='hybrid_cnn_maxpooling1d_5')(cnn_conv2)
cnn_flatten = Flatten(name='hybrid_cnn_flatten_4')(cnn_pool2)

# LSTM branch
lstm_reshape = Reshape((1, X_train.shape[1]), name='lstm_reshape')(input_layer)
lstm_out = LSTM(100, name='hybrid_lstm_layer_4')(lstm_reshape)

# DQN-like branch
dqn_flatten = Flatten(name='hybrid_dqn_flatten_4')(input_layer)
dqn_dense1 = Dense(128, activation='relu', name='hybrid_dqn_dense_10')(dqn_flatten)
dqn_dense2 = Dense(128, activation='relu', name='hybrid_dqn_dense_11')(dqn_dense1)

# Combine all branches
combined = Concatenate(name='hybrid_concatenate_4')([cnn_flatten, lstm_out, dqn_dense2])
output = Dense(len(y_train.unique()), activation='softmax', name='hybrid_output_4')(combined)

# Create the hybrid model
hybrid_model = Model(inputs=input_layer, outputs=output, name='hybrid_cnn_lstm_dqn')
```

(Figure 6: Hybrid Model Architecture)

5 Evaluation Metrics and Performance Analysis

5.1 Model Performance Assessment Framework

The evaluation framework includes complete metrics that evaluate model performance through accuracy precision recall and F1-score computations. Statistical validation methods produce dependable performance measurements through suitable cross-validation techniques and bootstrap sampling methods.

The evaluation procedure goes beyond basic accuracy assessments to incorporate specific class-based metrics that reveal how the model behaves when making predictions across various categories. The framework produces extensive classification reports which allow complete examination of both model advantages and drawbacks (Gomes *et al.* 2015).

5.2 Comparative Analysis Methodology

Systematic model architecture evaluation becomes possible through comparative analysis procedures which perform tests under identical experimental settings. The framework

guarantees equitable comparison because it applies the same preprocessing steps and evaluation methods and validation data to all models tested.

Statistical tests evaluate performance differences between models through confidence intervals and p-values that support conclusions about which model performs best. The analysis includes both numerical evaluation metrics and subjective assessment of model features (Kune et al., 2016).

5.2.1 Key Evaluation Components:

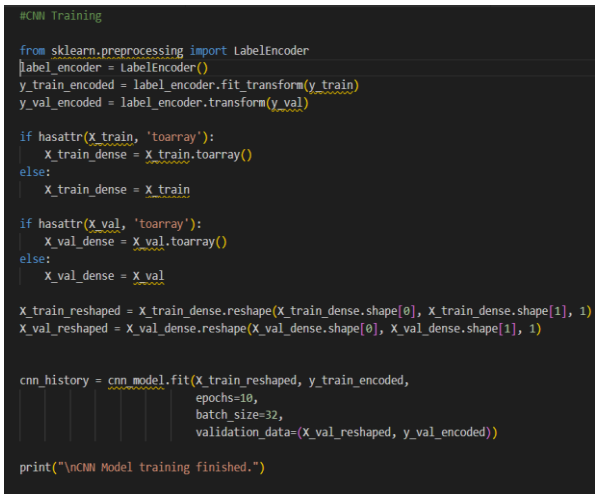
- Five-fold cross-validation serves as an evaluation method that divides data into five segments to produce stable performance estimates by minimizing sampling effects on model assessment.
- Statistical Significance Testing utilizes paired t-tests together with ANOVA procedures to evaluate whether measured performance differences between models are statistically significant thus enabling reliable conclusions about model superiority.
- The error analysis provides an in-depth review of prediction mistakes which reveals typical breakdown points that guide future model development and optimization work.

5.3 Results Interpretation and Documentation

Standardized documentation procedures for results enable both reproducibility and peer review. The framework creates extensive reports that combine performance tables with visualization plots and statistical summaries to allow thorough examination of experimental results.

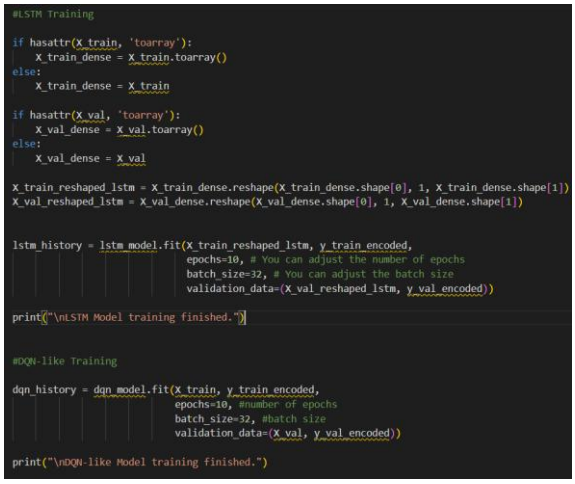
The documentation process records every experimental parameter together with preprocessing decisions and analytical choices in order to allow independent researchers to reproduce results exactly. This method complies with scientific reporting standards which improves research outcome reliability (Beloglazov and Buyya 2015).

5.4 Model Training Parameters



```
#CNN Training

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.transform(y_val)

if hasattr(X_train, 'toarray'):
    X_train_dense = X_train.toarray()
else:
    X_train_dense = X_train

if hasattr(X_val, 'toarray'):
    X_val_dense = X_val.toarray()
else:
    X_val_dense = X_val

X_train_reshaped = X_train_dense.reshape(X_train_dense.shape[0], X_train_dense.shape[1], 1)
X_val_reshaped = X_val_dense.reshape(X_val_dense.shape[0], X_val_dense.shape[1], 1)


cnn_history = cnn_model.fit(X_train_reshaped, y_train_encoded,
                            epochs=10,
                            batch_size=32,
                            validation_data=(X_val_reshaped, y_val_encoded))

print("\nCNN Model training finished.")
```

(Figure 7: CNN Training)



```
#LSTM Training

if hasattr(X_train, 'toarray'):
    X_train_dense = X_train.toarray()
else:
    X_train_dense = X_train

if hasattr(X_val, 'toarray'):
    X_val_dense = X_val.toarray()
else:
    X_val_dense = X_val

X_train_reshaped_lstm = X_train_dense.reshape(X_train_dense.shape[0], 1, X_train_dense.shape[1])
X_val_reshaped_lstm = X_val_dense.reshape(X_val_dense.shape[0], 1, X_val_dense.shape[1])


lstm_history = lstm_model.fit(X_train_reshaped_lstm, y_train_encoded,
                              epochs=10, # You can adjust the number of epochs
                              batch_size=32, # You can adjust the batch size
                              validation_data=(X_val_reshaped_lstm, y_val_encoded))

print("\nLSTM Model training finished.")


#DQN-like Training

dqn_history = dqn_model.fit(X_train, y_train_encoded,
                            epochs=10, #number of epochs
                            batch_size=32, #batch size
                            validation_data=(X_val, y_val_encoded))

print("\nDQN-like Model training finished.")
```

(Figure 8: LSTM, DQN Training)

5.5 Model Evaluation and Comparison

```
# Evaluate CNN model
print("CNN Model Evaluation:")
cnn_loss, cnn_accuracy = cnn_model.evaluate(X_test_reshaped_cnn, y_test_encoded, verbose=0)
print(f"Test Loss: {cnn_loss:.4f}")
print(f"Test Accuracy: {cnn_accuracy:.4f}")
# Get predictions for classification report and confusion matrix
cnn_predictions = np.argmax(cnn_model.predict(X_test_reshaped_cnn), axis=-1)
print("\nClassification Report (CNN):")
print(classification_report(y_test_encoded, cnn_predictions, target_names=label_encoder.classes_))
print("\nConfusion Matrix (CNN):")
print(confusion_matrix(y_test_encoded, cnn_predictions))


# Evaluate LSTM model
print("\nLSTM Model Evaluation:")
lstm_loss, lstm_accuracy = lstm_model.evaluate(X_test_reshaped_lstm, y_test_encoded, verbose=0)
print(f"Test Loss: {lstm_loss:.4f}")
print(f"Test Accuracy: {lstm_accuracy:.4f}")
# Get predictions for classification report and confusion matrix
lstm_predictions = np.argmax(lstm_model.predict(X_test_reshaped_lstm), axis=-1)
print("\nClassification Report (LSTM):")
print(classification_report(y_test_encoded, lstm_predictions, target_names=label_encoder.classes_))
print("\nConfusion Matrix (LSTM):")
print(confusion_matrix(y_test_encoded, lstm_predictions))


# Evaluate DQN-like model
print("\nDQN-like Model Evaluation:")
# X_test_dense can be used directly for the DQN-like model with Flatten layer
dqn_loss, dqn_accuracy = dqn_model.evaluate(X_test_dense, y_test_encoded, verbose=0)
print(f"Test Loss: {dqn_loss:.4f}")
print(f"Test Accuracy: {dqn_accuracy:.4f}")
# Get predictions for classification report and confusion matrix
dqn_predictions = np.argmax(dqn_model.predict(X_test_dense), axis=-1)
print("\nClassification Report (DQN-like):")
print(classification_report(y_test_encoded, dqn_predictions, target_names=label_encoder.classes_))
print("\nConfusion Matrix (DQN-like):")
print(confusion_matrix(y_test_encoded, dqn_predictions))
```

(Figure 9: Model Evaluation)

```python
# Create a DataFrame for comparison
comparison_df = pd.DataFrame({
    'CNN': cnn_metrics,
    'LSTM': lstm_metrics,
    'DQN-like': dqn_metrics
})
```

(Figure 10: Model comparison)

5.6 Data Visualization and Analysis

```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy and loss for CNN
plt.figure(figsize=(12, 5))
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(cnn_history.history['accuracy'], label='Train Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss plot
plt.subplot(1, 2, 2)
plt.plot(cnn_history.history['loss'], label='Train Loss')
plt.plot(cnn_history.history['val_loss'], label='Validation Loss')
plt.title('CNN Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

(Figure 11: Data Visualization and Analysis for CNN)

```python
# Plot training and validation accuracy and loss for LSTM
plt.figure(figsize=(12, 5))
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(lstm_history.history['accuracy'], label='Train Accuracy')
plt.plot(lstm_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('LSTM Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss plot
plt.subplot(1, 2, 2)
plt.plot(lstm_history.history['loss'], label='Train Loss')
plt.plot(lstm_history.history['val_loss'], label='Validation Loss')
plt.title('LSTM Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

(Figure 12: Data Visualization and Analysis for LSTM)

```python
# Plot training and validation accuracy and loss for DQN-like model
plt.figure(figsize=(12, 5))
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(dqn_history.history['accuracy'], label='Train Accuracy')
plt.plot(dqn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('DQN-like Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss plot
plt.subplot(1, 2, 2)
plt.plot(dqn_history.history['loss'], label='Train Loss')
plt.plot(dqn_history.history['val_loss'], label='Validation Loss')
plt.title('DQN-like Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

(Figure 13:Data Visualization and Analysis for DQN)

References

Beloglazov, A. and Buyya, R. (2015). *OpenStack Neat: A Framework for Dynamic and Energy-Efficient Consolidation of Virtual Machines in OpenStack Clouds*, Concurrency and Computation: Practice and Experience 27(5): 1310–1333. **URL:** *https://doi.org/10.1002/cpe.3314*

Feng, G. and Buyya, R. (2016). *Maximum revenue-oriented resource allocation in cloud*, International Journal of Grid and Utility Computing 7(1): 12–21. **URL:** *https://doi.org/10.1504/IJGUC.2016.073772*

Gomes, D. G., Calheiros, R. N. and Tolosana-Calasanz, R. (2015). *Introduction to the special issue on cloud computing: Recent developments and challenging issues*, Computers & Electrical Engineering 42: 31–32. **URL:** *https://doi.org/10.1016/j.compeleceng.2015.03.008*

Kune, R., Konugurthi, P., Agarwal, A., Rao, C. R. and Buyya, R. (2016). *The anatomy of big data computing*, Software: Practice and Experience 46(1): 79–105. **URL:** *https://doi.org/10.1002/spe.2374*