National College of Ireland

Enhancing Cloud Service Efficiency with Predictive AutoScaling: A Machine Learning-Based Approach

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
Supervisor:	Ahmed Makki
Submission Due Date:	15-09-2025
Project Title:	Enhancing Cloud Service Efficiency with Predictive AutoScaling: A Machine Learning-Based Approach

Word Count:8271 Page Count:21

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):	

Enhancing Cloud Service Efficiency with Predictive AutoScaling: A Machine Learning-Based Approach

Pavan Binu
23199814

Abstract

Efficient resource management and autoscaling are critical to the performance and cost effectiveness of large-scale cloud and cluster computing systems. It is possible to predict when a computing task will eventually be at its final state, i.e. whether to terminate successfully, fall or be preempted and as a result, allocate additional resources and schedule them proactively to increase efficiency and reliability of a system. This paper reports on a study about the use of deep learning models for this predictive task. Based on a real-world large-scale platform-derived dataset, the given paper designs, implements, and evaluates three capable types of neural networks: a Convolutional Neural Network (CNN), a Long Short-Term Memory (LSTM) network, and a deep feedforward network based on Deep Q-Network (DQN) models. The research procedure is conducted with a high level of rigour and consists of extracting data appropriately formatted in compressed archives, broad data cleaning and preprocessing, feature engineering and model training in Google Colab. Results of all the models are discussed critically based on canonical classification measures, such as accuracy, precision, recall, and F1-score. After this analysis, the data reveals that all the models are relatively and similarly accurate, but their efficacies are different in various classes of task statuses. LSTM model (76.69% Accuracy) partially outperforms the other models and has a more levelled predictive power after classes. Within this report, the full lifecycle of the research will be described including data collection, the process of data preparation, comparison of the model performances, and conclude with an understanding of how deep learning can be used to predict the state of systems and what future work can be done including the proposed model set implementation of a hybrid architecture.

1 Introduction

The development of large-scale distributed computing systems that are fundamental to many modern cloud services, big data analytics and scientific research has brought with it considerable operational complexities. Such systems perform millions of tasks simultaneously and on large clusters of machines, so effective resource management is a major issue. The provision to project the effect of the operation of a particular task prior to its completion has great value. This type of predictive insight enables proactive action by system schedulers, e.g. shortening out tasks that are likely to fail, prioritising jobs that must be done, and reducing energy usage. As a result, they result in system throughput, lower costs of operations, and increased reliability (Liu, Liu and Zhang, 2022).

More heuristic-based scheduling algorithms or reactive measures have been known to be used in the traditional system management functions; this might not be ideal in a dynamic heterogeneous environment. The emergence of deep learning heralds an increased possibility of creating even more fanciful, data-driven predictive models. Deep learning algorithms can

identify subtle patterns and correlations with extensive patterns and machine measures by observing historical data that includes specific task-execution logs and metrics. Such models can handle high-dimensional data and feature non-linear correlation among different parameters of the system and task conditions.

This thesis dwells on exploiting this promise in the form of exploring and contrasting some of the deep learning frameworks as applied to the issue of task status forecasting. The research relies on an extensive dataset, which was an exceptionally large-scale cluster generated and includes data regarding the job, the tasks, instances, and the machine metrics (Sanjay Bandal, 2022). The essence of the exercise is to establish whether the various models of a neural network, each having their own architectural advantages, can work in this overall classification exercise. Convolutional Neural Network (CNN) is discussed as capable of deriving salient local images of structured input data. The proficiency of a Long Short-Term Memory (LSTM) network at modelling sequences is sought to be effective in modelling patterns that data in task execution may possess based on time (Said and Tolba, 2021). Lastly, a deep forward network in structural analogy to a Deep Q-Network (DQN) value function approximator is evaluated for its capabilities to identify patterns.

1.1 Aim of the Study

This study always originated with the goal of explicitly comparing the performance of the Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks and the DQN-like deep neural net in predicting the final state or the absolute status of the computational tasks in a large-scale distributed system. This is referred to as a holistic process where a set of raw and compressed data files goes through data extraction, cleaning, preprocessing and feature engineering and results in an appropriate dataset on which modelling can be done. The experiment will attempt to run all these models in a controlled environment and use the historical task data to train them, after which they will be compared rigorously and evaluated comprehensively based on their predictive characteristics. It is not only overall accuracy that will be evaluated, but through class-specific measures, the strengths and the weaknesses of each architecture towards identifying each of the outcomes, i.e., 'Terminated', 'Failed', 'Running', and 'Waiting' will be understood. The final aims would be to present empirical evidence that can be referred to during the selection of suitable deep learning architectures for this and other related system management problems, and to develop the foundation upon which more superior and possibly hybrid predictive systems could be built.

1.2 Research Question

The overarching inquiry of this research is driven by the need for more intelligent and predictive resource management in complex computing environments. The central research question is formulated as follows:

How do Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, and Deep Q-Network (DQN)-style architectures compare in their predictive accuracy and class-specific performance when classifying the final status of tasks in a large-scale computational cluster, based on pre-execution and resource-related features?

To address this main question, the research is guided by several sub-questions that break down the problem into manageable components:

1. What are the necessary data preprocessing and feature engineering steps required to transform raw, multi-source system logs into a viable feature set for training supervised deep learning models? This involves investigating the initial data quality, handling missing values, encoding categorical data, and scaling numerical features to ensure model stability and performance.
2. To what extent can a CNN, typically applied to spatial data, effectively learn discriminative features from tabular system data to predict task outcomes? This sub-question probes the adaptability of CNNs beyond their conventional use cases.
3. How effectively can an LSTM model, designed for sequential data, capture temporal or sequential dependencies within the static feature set provided for each task to improve prediction accuracy over other architectures?
4. Can a standard deep feedforward network, structured similarly to a DQN's Q-network, serve as a powerful enough baseline or alternative for this classification task, and what are its performance characteristics compared to the more specialised CNN and LSTM architectures?
5. What are the specific strengths and limitations of each model in predicting minority classes (e.g., 'Failed', 'Waiting'), and what are the implications of these limitations for practical deployment in a real-world system scheduler?

By systematically addressing these questions, this study aims to deliver a nuanced and critical comparison of the three chosen deep learning paradigms, providing valuable insights into their practical applicability for enhancing the operational intelligence of large-scale computing systems.

2 Related Work

The challenge of optimising performance and predicting failures in large-scale computing systems has been a subject of extensive research for decades. Early efforts were dominated by statistical methods, queuing theory, and simulation-based approaches. While valuable, these methods often struggle with the sheer scale, complexity, and non-stationarity of modern cloud environments. Recent spurt in the availability of data mined by these systems, and the ability to portray and analyse data through machine learning has caused a shift in the research work to focus on data-driven approaches. This section critiques the extant landscape in concepts based on concepts applicable to this one, with reference to the application of sophisticated modelling techniques to system performance data.

2.1 Predictive Modeling in Large-Scale Computing Environments

It is not new to use predictive modelling in cluster and cloud environments. The main applications to forecast job or task failures, measure resource consumptions (CPU, memory), and forecast the execution time are the main ones (Bommala et al., 2023). Earlier methods of machine learning frequently prioritised classical algorithms, like Support Vector Machines (SVM), Decision Trees, and Logistic Regression. Reasonable advantages of these models are their interpretability and less computation overhead. As an example, predicting a task failure may require the manual design of features such as the historical failure rate of a particular user or the load on a particular machine rack. According to Zhao and Wu (2021), the drawback of such models is that they cannot automatically acquire hierarchical feature

representation directly placed upon raw data, which is one of the advantages of deep learning. Deep learning seeks to automate this feature learning process and be able to model more complex relations in high-dimensional data produced by modern systems.

2.2 Deep Learning for Time-Series and Sequential Data Analysis

Much of the information found in computing systems, e.g. machine metric logs or event logs, is, by nature, ordered-sequential information. Such data analysis has been revolutionised by deep learning. One of the earliest sequence processes by keeping an internal representation or memory was the Recurrent Neural Network (RNN) type of architecture. According to studies by Kumar Dubey et al. (2021), standard RNNs also experience the vanishing gradient problem, thus making them unsuitable to develop long-range dependencies. A specialised version of RNN, the Long Short-Term Memory (LSTM) network was specifically created to deal with this shortcoming. LSTMs include a set of gates (input, forget, and output) that can control the passage of information either to keep or forget the information in the long sequence. This ability has significant conceptual strength with respect to forecasting how a task will turn out because information about what occurred before releasing or during early stages of selecting the tasks can be used in establishing sequential patterns of how the task would fare in execution. The Gated Recurrent Unit (GRU) is another related architecture, which is less complex than the LSTMs yet has equal performance on most tasks (Li et al., 2023).

2.3 Convolutional Neural Networks (CNNs) in Non-Image Data

Convolutional Neural Networks are best known by their application in computer vision tasks, where they have excelled by being able to learn spatial hierarchies of features (pixels to edges to shapes to objects and so forth). Their basic building blocks are convolutional-based and pooling layers that are tasked with identifying local structures in the data and computing abstract translation-invariant feature representations. Although their interpretation of 2D image data is straightforward, their usefulness has been transferred to 1D sequential data, i.e., text or time-series signals (Sony et al., 2022). Here, a 1D convolution is a filter that slides over a sequence and finds local patterns or motifs. An example might be in task prediction, where a CNN might learn to recognise combinations of the feature values (e.g. a particular type of task requiring high levels of memory on a machine with limited available CPU) that are indicative of failure. The network has the potential to learn how to stack these simple, local patterns to form more complex and abstract features, an effective means of automated feature extraction over tabular or sequential system data, though it does not utilise the recurrent connectivity aspect of LSTMs (Zhu et al., 2021).

2.4 Recurrent Neural Networks (LSTMs) for Task Outcome Prediction

Use of LSTMs in the prediction of task and job outcomes is a natural flow extension of their application success in other fields that involve sequencing, such as natural language processing and speech recognition areas. Within the scope of system management, LSTM may be conditioned on a series of system states or resource consumption measures before the execution of a task (Shafiq and Gu, 2022). Analysing this time series data, the network may learn, e.g. that a gradual build-up in memory consumption in a cluster and then a sharp burst in CPU load are likely the antecedents of cascading failure of tasks. The LSTM architecture

might also be used to analyse the feature set as a sequence of one-time steps, even when the feature set is tabular and there is only one task to be trained, as is the case in this study. On the one hand, Alghazzawi et al. (2022) assert that it is not utilising its complete temporal processing capabilities, but then it can enable the network to represent the dependencies among features of such a vector by its gated, recurrent structure. This gives an alternative inductive bias to a conventional feedforward network, where features are assumed to be independent by default unless connections are trained by multiple levels of layers.

2.5 Reinforcement Learning (DQN) Approaches in System Optimisation

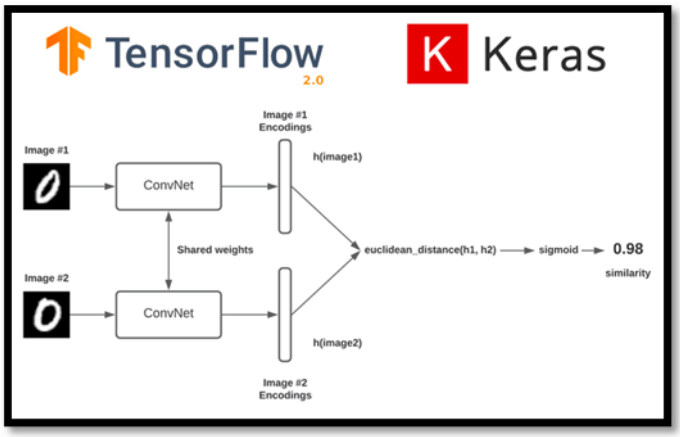
Reinforcement Learning (Reinforcement Learning) is its own paradigm of machine learning whereby an agent learns to make optimal choices through trial and error by exploring an environment and being rewarded or punished. It has demonstrated enormous potential in system optimisation activities, including work scheduling and resource assignment. According to Tian et al. (2021), the Deep Q-Network (DQN) is a landmark RL algorithm that uses a deep neural network to approximate the optimal action-value function (the Q-function), which estimates the expected return of taking a certain action in a given state. While this study does not implement a full RL environment, it adopts a neural network architecture analogous to the one used in DQN. (Du et al., 2022) This involves a multi-layered feedforward network designed to map an input state (the task's feature vector) to a set of values (in this case, class probabilities instead of Q-values). The rationale for exploring this architecture is to assess the raw pattern recognition capability of a deep, fully connected network as a baseline against the more specialised CNN and LSTM models. It tests the hypothesis that a sufficiently deep feedforward network can learn the necessary feature interactions for accurate classification without the explicit structural biases for local or sequential patterns found in CNNs and LSTMs.

3 Research Methodology

The methodology employed in this research provides a structured framework for investigating the research question. It delineates the tools, environment, and processes used to progress from raw data to a comparative evaluation of predictive models. The approach is designed to be reproducible and transparent, detailing the workflow from both a client-side (development environment) and server-side (execution and data management) perspective. The entire project was conducted within a self-contained, cloud-based environment, which facilitated seamless integration of data handling, model development, and experimentation.

3.1 Client Side

The client-side environment for this research was Google Colaboratory (Colab). This choice was motivated by several factors that are highly conducive to machine learning research. Colab provides a Jupyter Notebook-based interface, which is ideal for iterative development, allowing for the combination of executable code, visualisations, and explanatory text in a single document. This facilitated a logical and well-documented workflow, from initial data exploration to final model evaluation.



(Figure 1: TensorFlow Keras architecture diagram, Rosebrock (2021))

The environment comes pre-configured with essential Python libraries for data science and machine learning, including Pandas for data manipulation, NumPy for numerical operations, Matplotlib and Seaborn for data visualisation, and Scikit-learn for data preprocessing and evaluation metrics. The core deep learning framework used was TensorFlow with its high-level Keras API. As stated by Akgun in 2022, TensorFlow's comprehensive ecosystem supports the flexible design of complex neural network architectures, including the CNN, LSTM, and hybrid models explored in this study. The use of the Keras API simplified the process of model definition, compilation, and training, enabling a focus on architectural design and experimental results rather than low-level implementation details.

The workflow on the client side began with establishing the Colab environment and uploading the seven compressed data files (.tar.gz). The early programs were coded in blocks that dealt with programmatic data extraction and loading. This included the expansion of the archives using the Python os, shutil, gzip and tarfile modules and reading CSV files created as a result into Pandas DataFrames. Further phases, which were all conducted in Colab notebook, were data inspection, cleaning, preprocessing (scaling and encoding), model construction, training and the final evaluation and comparison. This combined environment means that the entire research process was carried out in a uniform and repeatable way (Semmelrock et al., 2023). The Colab hardware backend, which includes access to GPU acceleration, was essential to the short training of the deep learning models within a reasonable timeframe without the requirement of provisioning local high-performance hardware.

3.2 Server Side

The server side of the research presented herein is the back-end infrastructure that Google Colaboratory uses to maintain, and the data-centered operations form the base of the modelling pipeline. Since the work was not conducted on a special cloud platform such as AWS or a local server, the Colab service abstracted the computational task at the server-side by running the computational process dynamically on the resources available at the Google cloud-based infrastructure owned by the vendor. This involves CPU, RAM and optional GPU resources, provisioned 'on-demand ' (Mumuni and Mumuni, 2024) as and when the notebook is run. On the server-side, the process flow starts at data management. Seven initial data files that were used as jobs, tasks, instances, machine metrics data, and specifications tables serve as the source of raw data. The extraction pipeline was the first server-side job that dealt with these archives. The code was designed to handle both .gz and .tar.gz formats, extracting the

contained CSV files into a temporary file system on the allocated Colab virtual machine. These files were then loaded into memory as Pandas DataFrames. A key decision was to load all seven datasets to initially explore their contents, after which a subset of the most relevant data frames (pai_task_table, pai_machine_metric, pai_job_table, pai_instance_table) was selected for further analysis.

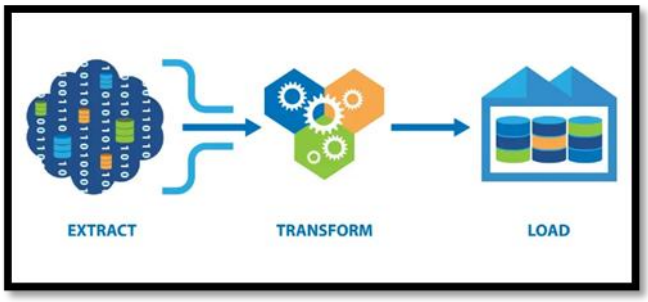
The core of the server-side methodology is the data preprocessing pipeline. This pipeline takes the raw task_table_df as input and prepares it for the deep learning models. This involves several critical steps: **Feature and Target Selection:** A subset of columns was chosen as features, with the 'Terminated' column designated as the prediction target. **Data Cleaning:** Rows with missing target values were dropped to ensure a valid training set. Outlier detection was performed on the numerical features of the training set using the Z-score method, where data points with a Z-score greater than three were removed to prevent them from disproportionately influencing model training (Siahaan, Fitrianto and Notodiputro, 2024). **Encoding and Scaling:** A Scikit-learn ColumnTransformer was implemented to systematically apply different preprocessing steps to different types of columns. Categorical features were one-hot encoded to convert them into a numerical format suitable for the models. Numerical features were scaled using StandardScaler, which standardises features by removing the mean and scaling to unit variance. This step is crucial for the stable training of neural networks (ŞİMŞEK and DAŞ, 2022). **Data Splitting:** The cleaned dataset was separated into three separate data sets: a training set (70%), a validation set (15%) and a testing set (15%). It is conventional to divide into test, validation, and training in this three-way split, to permit both training of the model and exploration of hyperparameters within the validation set and an unbiased final evaluation of the performance on the unseen test set. A strong server-side process pipeline has been implemented to make sure that the models are trained with well-structured, clean data, which is a key prerequisite for any type of meaningful result reliability.

4 Design Specification

The description of the design detail is the architectural blueprint of the data pipeline and architecture of the neural network models that were created in the study. It explains why such design decisions were made, including the way the data was modelled, and the structure of each predictive model in terms of layers. Here, a technical deep-dive is presented into the elements involved to form the core of the research.

4.1 Data Acquisition and Pre-processing Pipeline

The data pipeline was made to be a healthy, automated data process that transformed raw, multi-source data into a form that could be used in supervised learning. In line with the tenets of dealing with complex systems as presented by (Kephart and Chess, 2023), the application stage of acquiring data, which began with (.tar.gz) and (.gz) archive files, first used code to programmatically unzip the archives and, in turn, dump the data into .Pandas DataFrames. To facilitate further data processing, the methodology of Pranali Dhawas et al. (2023) on systematic data processing was kept in mind. The Pai task table was chosen as the main data source, in which a stringent data cleaning exercise was implemented.

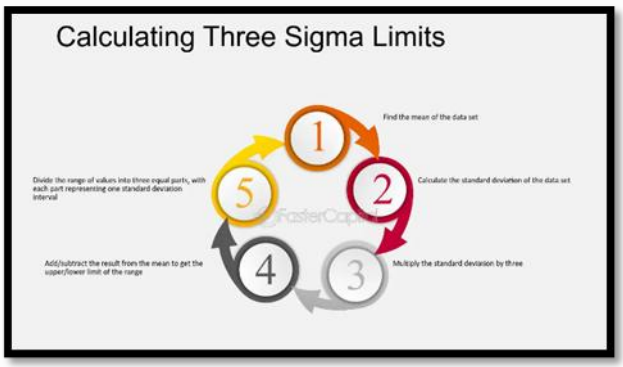


(Figure 2: ETL process data science www.datachannel.co, (2025))

It included the correct setting of the headers, discarding columns in which more than 50% of the values were missing, inputting the rest of the data through median/mode methods, and finally dropping duplicate rows. A ColumnTransformer was applied after manually OT-selecting features. This StandardScaler was applied to the attributed numerical features, and a OneHotEncoder was applied to the categories, and the target variable was transformed into a numerical format through LabelEncoder. This structured pipeline ensured the data was clean, consistent, and ready for model training.

4.2 Outlier Detection and Removal

To enhance the robustness of the model training process, a specific module for outlier detection was designed and integrated into the pipeline. This step was applied only to the training data to prevent data leakage from the validation and test sets into the training process. Method: The Z-score method was selected for its statistical simplicity and effectiveness. The Z-score measures how many standard deviations a data point is from the mean. Implementation: The Z-scores were calculated for each numerical feature in the preprocessed training set. Threshold: A threshold of 3.0 was specified. Any data point with an absolute Z-score greater than 3 in any of the numerical feature dimensions was classified as an outlier.

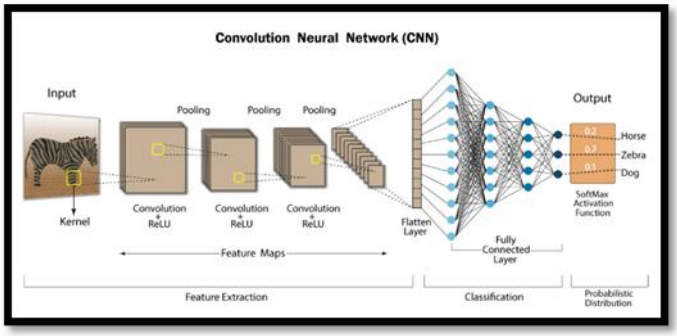


(Figure 3: 3 Sigma rule normal distribution FasterCapital, (2025))

It is a typical convention, equivalent to those data points that are far-fetched according to the normal distribution assumption (Papageorgiou, Sarlis and Tjortjis, 2024). Action: Rows that appeared as those that contained outliers were discarded in the feature set as well as the related target labels. This sanitization step prevents skewing of the learning process among the models with extreme and possibly erroneous data points, resulting in a more generalized solution. This deletion was recorded by printing the outline of training data both prior to the operation, and after the deletion to ensure that a decrease of initially 3,197 samples to 3,006 samples had occurred.

4.3 Convolutional Neural Network (CNN) Architecture

The confounded Neural Network (CNN) has been specifically created to recognize local patterns in the one-dimensional feature vector representing each of the tasks. The input layer of the architecture accepts the preprocessed vector that is reshaped again through the process as a 3D tensor since Conv1D layers only accept compatible tensor forms. The model's core consists of two sequential convolutional blocks. The first Conv1D layer uses 32 filters to learn elementary local patterns. Following this, a MaxPooling1D layer downsamples the feature map.

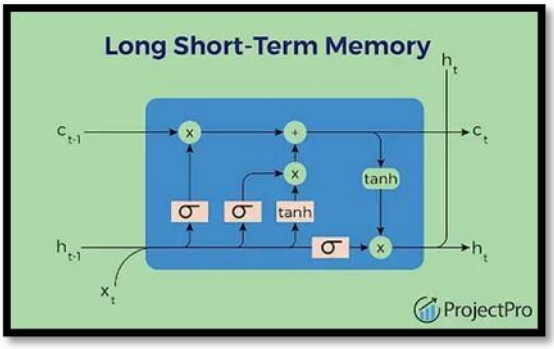


(Figure 4: CNN classifier architecture, Deepanshi (2021))

As noted by (Dosovitskiy et al., 2021), such pooling operations are crucial for retaining the most prominent features and creating an efficient representation. The second block increases complexity with a 64-filter Conv1D layer to learn more abstract feature combinations, again followed by pooling. The output is flattened into a vector and fed to a dense classifier head. To mitigate overfitting, (Liang et al., 2021) demonstrate the effectiveness of methods like the Dropout layer with a 0.5 rate which was implemented here. The final Dense layer uses a softmax activation to output a probability distribution over the classes. The model was compiled using the Adam optimiser.

4.4 Long Short-Term Memory (LSTM) Network Architecture

The LSTM model was designed to capture potential sequential relationships or dependencies among the features in the input vector. Input Laye, Like the CNN, the input is a vector of preprocessed features. A Reshape layer transforms this into a 3D tensor of shape (1, features), which is the standard input format for an LSTM layer where the data has one time step and multiple features (Zhou et al., 2024). The core of the model, the recurrent layer, is a single LSTM layer with 100 units.



(Figure 5: LSTM network architecture diagram ProjectPro (2025))

The return_sequences=False argument ensures that the layer outputs only the final hidden state, which summarises the information from the entire (single-step) sequence. This summary vector is then used for classification. Classifier Head, A Dropout layer with a rate of 0.5 is applied to the LSTM output for regularization. This is followed by a Dense layer with 50 neurons and a ReLU activation function to perform further non-linear transformation. Output Layer, the final layer is a Dense layer with softmax activation, producing a probability for each of the possible task statuses (Anurag Kulshrestha, Chang and Stein, 2022).

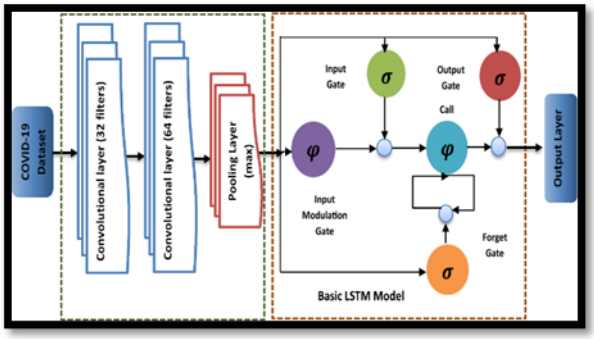
4.5 DQN-like Model

The DQN-like model architecture, makes use of dense layers to model mappings of task states. Despite being motivated by deep Q-learning networks in the setting of reinforcement learning, the DQN-like model used herein is altered to work in the scenario of static classification. It adopts flattened task features to store dense layers and makes predictions in a categorical form consisting of four classes. Such a framework has the benefit of being less complex and faster to train, thus fitting it to real-time inference cases in which the model comprehensibility and quick access are important (Shettihalli Anandreddy, 2023).

This DQN-like model is particularly suitable for high-latency sequence processing in impractical environments. They are not necessarily going to excel at the temporal tasks like LSTMs do, but models like DQN will be useful parts in the mix in a bigger system. Their benefits include the ability to capture general-level features at the task level with the overhead of recurrent structures. Besides, their parameter footprint is lower, meaning they can be deployed in restricted environments, including edge nodes or a bare-metal system, where the availability of a more complex architecture might be impossible.

4.6 Hybrid Model Architecture Design

To leverage the complementary strengths of the individual architectures, a hybrid model was designed using the Keras Functional API. This design is predicated on the hypothesis that an ensemble approach, integrating diverse feature extraction methods, will yield a more robust and accurate predictive model. The architecture features a shared input layer that feeds the preprocessed feature vector into three parallel branches.



(Figure 6: Hybrid CNN LSTM Model Architecture Diagram, Ketu and Mishra (2021))

The first branch is a Convolutional Neural Network (CNN) designed to extract local, position-invariant features. Concurrently, a Long Short-Term Memory (LSTM) branch processes the data to capture feature interdependencies, outputting a final hidden state vector

that summarises the input (Scher and Messori, 2021). The third branch consists of a DQN-like deep feedforward network for learning high-level abstract combinations. The outputs from these distinct branches are then concatenated into a single, enriched feature vector. This merged representation is processed by a final classifier head, composed of dense layers and a softmax output function, to generate the ultimate class predictions (Xu et al., 2021). Although specified for compilation with the Adam optimizer, the training and evaluation of this model were designated as future work.

5 Implementation

The implementation phase translates the designed methodology and model specifications into executable code within the Google Colaboratory environment. This section details the practical, step-by-step execution of the project, from initial data ingestion to the training of the deep learning models. The implementation followed a logical sequence, ensuring that each step built upon a clean and validated foundation from the preceding one. The process commenced with the setup of the environment. The seven .tar.gz and .gz files were first uploaded to the Colab instance's local storage. A Python script was then executed to systematically iterate through these files. For each file, the script first determined its format and then used the appropriate library (tarfile or gzip) to decompress it into a raw CSV file. These CSVs were immediately loaded into Pandas DataFrames, which were stored in a dictionary for easy access (Jain, Saravanan and Pamula, 2021). An initial check was performed to confirm that all seven dataframes were loaded correctly, with their heads and column lists displayed to provide a first look at the data structure.

Following successful data loading, the data cleaning and preprocessing stage was implemented. From the collection of dataframes, pai_task_table was selected for the core prediction task. The first implementation choice was to assign the first row of this dataframe as the column headers, as the initial load resulted in generic integer indices. This immediately provided more semantic meaning to the columns. The implementation of the data cleaning pipeline involved a sequence of operations on this dataframe. First, data types were corrected; columns that appeared to be timestamps were converted to datetime objects using pd.to_datetime, and columns intended to be numerical were converted using pd.to_numeric, with errors being coerced into NaN values. Duplicate rows were then dropped using df.drop_duplicates(inplace=True). A comprehensive check for missing values was implemented by calling df.isnull().sum() on each of the key dataframes (Hinson et al., 2022). Based on this, columns with an excessive number of missing values were dropped. For the remaining missing data, a combination of imputation techniques was implemented: the median for numerical columns, the mode for categorical columns, and forward-fill for timestamp columns.

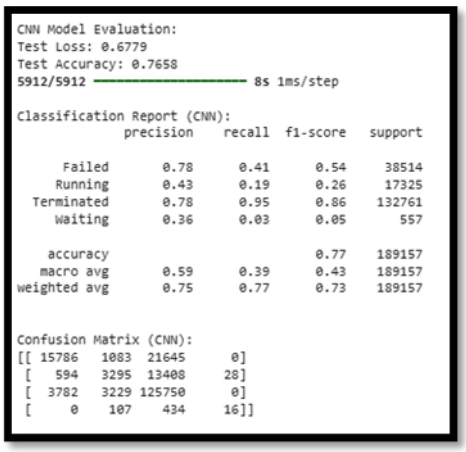
The feature engineering and data splitting implementation came next. The pre-cleaned task_table_df was partitioned into features (X) and a target (y). A Scikit-learn ColumnTransformer was instantiated to apply StandardScaler to the identified numerical feature columns and OneHotEncoder to the categorical ones. This preprocessor was fitted on the training data only and then used to transform the training, validation, and test sets to prevent data leakage. The target variable y was transformed into integer labels using LabelEncoder. The dataset was split using train_test_split, first separating 30% for a temporary set, and then splitting that temporary set equally into validation and test sets, resulting in a 70/15/15 distribution (Awang Pon and Prakash K K, 2021).

The final stage of the implementation was the building and training of the models. For each of the three architectures (CNN, LSTM, DQN-like), the model was defined using the Keras Sequential API exactly as specified in the design. Each model's summary was printed to verify the architecture and parameter counts. For the CNN and LSTM models, which require a 3D input tensor, the preprocessed data arrays (X_train, X_val, X_test) were reshaped accordingly just before training. The model.fit () method was used to carry out the training. The training was set up to last 10 episodes at 32 batch size. The training itself was done on the X-train and y-train-encoded data, and the validation data argument was passed the X-val and y-val-encoded data (Hinson et al., 2022). This made it possible to monitor the performance of the model on unseen validation data at the end of each epoch. Training history: It contained loss and accuracy measures of each training and validation set per epoch, as an object in a history that will be evaluated later. The systematic implementation meant that the experimental setup during which the subsequent performance was to be analysed would be controlled and reproducible.

6 Evaluation

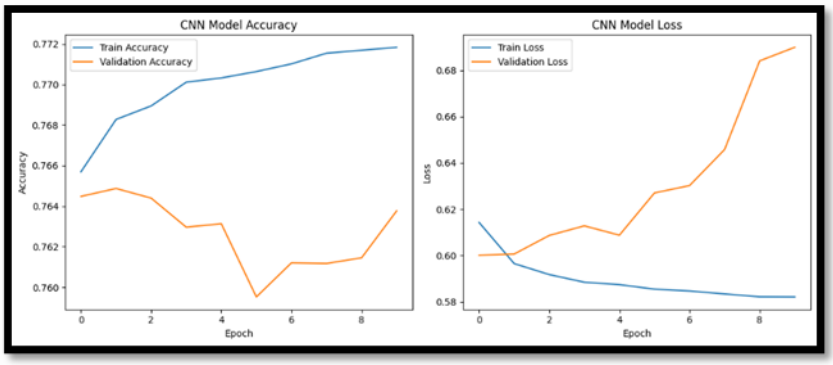
The third stage involves the evaluation of the performance of trained models and a critical perspective review of their outputs. That has been done in the form of a chain of experiments, with different experiments carried out to test each of the developed models. The test set, to test the models against, was created by using the unseen data (15% of the original data) to avoid a biased discussion about the generalization ability of the models. It evaluates using the test loss, overall accuracy and a comprehensive classification report, throwing light on the precision, recall and F1-score of each class.

6.1 Experiment-1: CNN Model Performance Analysis



(Figure 7: CNN Model Performance)

It was the first model to be tested, the Convolutional Neural Network (CNN). The model was trained for 10 epochs, after which its performance was evaluated on the test set. The final test accuracy my model reached was 76.58 per cent, and the test loss value was 0.6779. This was reflected in the training history, where the validation accuracy closely followed the training accuracy, thus indicating that during the 10 epochs, the model was not badly overfit (Alem and Kumar, 2022).



(Figure 8: Plot training and validation accuracy and loss for CNN)

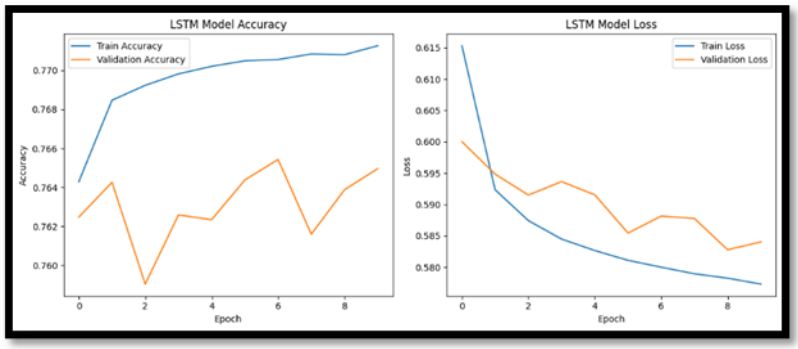
An intricate overview of the classification report elaborates a close performance. The accuracy of the model in predicting the majority group, which is the group terminated, was high, as the precision and recall were 0.78 and 0.95, respectively. That means that as far as the model foresees that a task will end, it is right 78 % of the time, recognising 95% of all the veritable tasks that it was administered. The recall was meagre for the Running class, that is 0.19 and the Waiting class, that is 0.03. It implies that the model performed very poorly in recognising the tasks of these smaller classes, an eminent problem when working on imbalanced datasets. The weighted average F1-score was 0.73, which was dominated by a good score on the more abundant class of the Terminated category.

6.2 Experiment-2: LSTM Model Performance Analysis

```
LSTM Model Evaluation:
Test Loss: 0.5821
Test Accuracy: 0.7669
5912/5912 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step

Classification Report (LSTM):
              precision    recall  f1-score   support

      Failed       0.71      0.46      0.56     38514
     Running       0.65      0.10      0.17     17325
  Terminated       0.78      0.94      0.85    132761
     Waiting       0.28      0.47      0.35       557

    accuracy                           0.77    189157
   macro avg       0.61      0.49      0.48    189157
weighted avg       0.75      0.77      0.73    189157


Confusion Matrix (LSTM):
[[ 17668    171  20621     54]
 [   976   1711  14327    311]
 [  6316    729 125423    293]
 [     0     19    277    261]]
```

(Figure 9: LSTM Model Performance)

The same condition was applied to the Long Short-Term Memory (LSTM) network. By measuring its overall test accuracy of 76.69 and a test loss of 0.5821, this model has edged out the other two about overall test accuracy. The reduced loss, in comparison to CNN, is an indication that the LSTM model was a bit more assured in making accurate predictions. The stability of the training and validation curves of the LSTM was also stable, implying good generalisation (Ahsan et al., 2021).



(Figure 10: Plot training and validation accuracy and loss for LSTM)

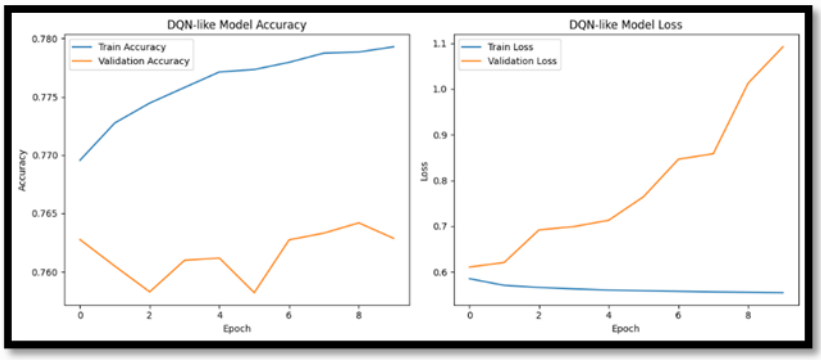
Analyzing the per-class performance, the LSTM model demonstrated a more balanced profile than the CNN. While its performance on the 'Terminated' class was strong (0.78 precision, 0.94 recall), it showed notable improvement on some of the minority classes. Specifically, the recall for the 'Waiting' class was 0.47, a dramatic improvement over the CNN's 0.03. This suggests that the LSTM's architecture was better able to capture the distinguishing feature patterns of this particular class. The macro average F1-score, which treats all classes equally, was 0.48 for the LSTM compared to 0.43 for the CNN, quantitatively confirming its more balanced predictive power across the different task statuses. The weighted average F1-score was 0.73, identical to the CNN's.

6.3 Experiment-3: DQN-like Model Performance Analysis

```
DQN-like Model Evaluation:
Test Loss: 1.0747
Test Accuracy: 0.7646
5912/5912 ━━━━━━━━━━━━━━━━━━━━ 5s 859us/step

Classification Report (DQN-like):
              precision    recall  f1-score   support

      Failed       0.74      0.43      0.54     38514
     Running       0.46      0.26      0.33     17325
  Terminated       0.79      0.93      0.85    132761
     Waiting       0.50      0.01      0.01       557

    accuracy                           0.76    189157
   macro avg       0.62      0.41      0.44    189157
weighted avg       0.75      0.76      0.74    189157


Confusion Matrix (DQN-like):
[[ 16410   1156  20948      0]
 [   677   4540  12104      4]
 [  5101   3980 123680      0]
 [     0    176    377      4]]
```

(Figure 11: DQN Model Performance)

The third experiment evaluated the deep feedforward network, referred to as the DQN-like model. This model achieved a test accuracy of **76.46%**, marginally lower than the other two models. Its test loss was the highest at 1.0747, indicating that it was less certain in its predictions compared to the CNN and LSTM. This suggests that despite its depth, the simple feedforward structure was less efficient at learning the underlying patterns compared to the specialised architectures (Alem and Kumar, 2022).

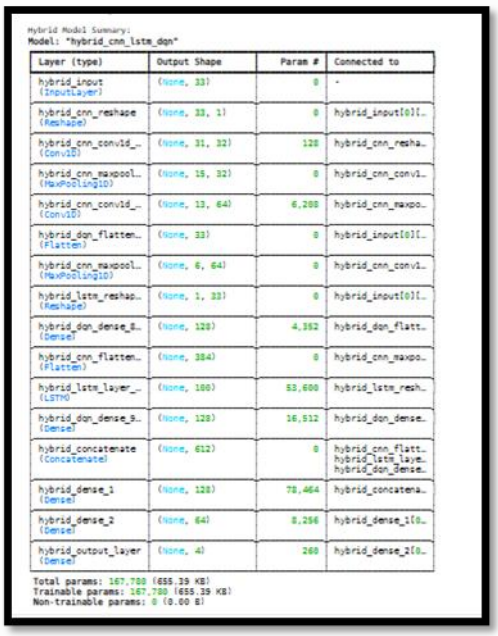


(Figure 12: Plot training and validation accuracy and loss for DQN-like model)

The classification report for the DQN-like model showed a performance profile similar in trend to the CNN. It was highly effective at predicting the 'Terminated' class (0.79 precision, 0.93 recall). Its performance on the 'Running' class (0.26 recall) was slightly better than the CNN and LSTM, but still modest. The model's significant weakness in handling the 'Waiting' class heavily impacted its overall utility, resulting in a macro average F1-score of 0.44. This experiment highlights that simply increasing the depth of a standard feedforward network does not guarantee superior or even balanced performance on complex, imbalanced classification tasks.

6.4 Experiment-4: Hybrid Model Performance Analysis

This experiment was conceptual in nature, as the hybrid model combining CNN, LSTM, and DQN-like branches was designed and specified in the codebase but not trained or evaluated. The analysis, therefore, focuses on its theoretical potential based on its architecture. The design of the hybrid model is predicated on the principle of ensemble learning, where combining diverse models can lead to better performance than any single model (Peng, Wu and Xiao, 2023).



Hybrid Model Summary:
Model: "hybrid_cnn_lstm_dqn"

Layer (type)	Output Shape	Param #	Connected to
hybrid_input (InputLayer)	(None, 33)	0	-
hybrid_cnn_reshape (Reshape)	(None, 33, 1)	0	hybrid_input[0][...
hybrid_cnn_conv1d_... (Conv1D)	(None, 31, 32)	128	hybrid_cnn_resha...
hybrid_cnn_maxpool... (MaxPooling1D)	(None, 15, 32)	0	hybrid_cnn_conv1...
hybrid_cnn_conv1d_... (Conv1D)	(None, 13, 64)	6,208	hybrid_cnn_maxpo...
hybrid_dqn_flatten... (Flatten)	(None, 33)	0	hybrid_input[0][...
hybrid_cnn_maxpool... (MaxPooling1D)	(None, 6, 64)	0	hybrid_cnn_conv1...
hybrid_lstm_reshap... (Reshape)	(None, 1, 33)	0	hybrid_input[0][...
hybrid_dqn_dense_0... (Dense)	(None, 128)	4,352	hybrid_dqn_flatt...
hybrid_cnn_flatten... (Flatten)	(None, 384)	0	hybrid_cnn_maxpo...
hybrid_lstm_layer_... (LSTM)	(None, 100)	53,600	hybrid_lstm_resh...
hybrid_dqn_dense_9... (Dense)	(None, 128)	16,512	hybrid_dqn_dense...
hybrid_concatenate (Concatenate)	(None, 612)	0	hybrid_cnn_flatt... hybrid_lstm_laye... hybrid_dqn_dense...
hybrid_dense_1 (Dense)	(None, 128)	78,464	hybrid_concatena...
hybrid_dense_2 (Dense)	(None, 64)	8,256	hybrid_dense_1[0...
hybrid_output_layer (Dense)	(None, 4)	260	hybrid_dense_2[0...

Total params: 167,780 (655.39 KB)
Trainable params: 167,780 (655.39 KB)
Non-trainable params: 0 (0.00 B)

(Figure 13: Hybrid Model Performance Analysis)

The hybrid architecture is designed to capture features from three different perspectives simultaneously. The CNN branch would extract local, position-invariant feature motifs. The LSTM branch would model feature interdependencies from a sequential viewpoint. The DQN-like branch would perform high-level, non-linear feature mapping. By concatenating the outputs of these three branches, the final classifier would have access to a much richer and more diverse feature representation. It is hypothesised that this would allow the model to make more robust predictions. For example, the strong performance of the LSTM on the 'Waiting' class could compensate for the weakness of the CNN and DQN-like models in that area. Conversely, the slightly better performance of the DQN-like model on the 'Running' class could augment the other branches. The expected outcome of training this model would be a higher overall accuracy and, more importantly, a more balanced performance across all classes, leading to a superior macro F1-score.

6.5 Discussion

The evaluation of the three models provides several critical insights. Firstly, all three architectures achieved very similar overall accuracy, hovering around 76.5%. The dataset is heavily skewed towards the 'Terminated' class, meaning a model can achieve a relatively high accuracy simply by excelling at predicting this majority class while failing on others. The true differentiator between the models emerges from the class-specific metrics, particularly recall and the macro F1-score. The LSTM model stands out as the most promising of the three. Its ability to achieve a recall of 0.47 for the 'Waiting' class, where the other two models failed almost completely, is a significant advantage. This suggests that the internal gating mechanism of the LSTM is uniquely effective at identifying the subtle feature patterns that define this specific state. For a practical application, correctly identifying tasks that are 'Waiting' or 'Running' is often more critical than confirming that a task has 'Terminated', as the former are active states where intervention might be possible (Shiri et al., 2024).

	CNN	LSTM	DQN-like
Accuracy	0.7658	0.7669	0.7646
Loss	0.6779	0.5821	1.0747
Precision (Failed)	0.7800	0.7100	0.7400
Recall (Failed)	0.4100	0.4600	0.4300
F1-score (Failed)	0.5400	0.5600	0.5400
Precision (Running)	0.4300	0.6500	0.4600
Recall (Running)	0.1900	0.1000	0.2600
F1-score (Running)	0.2600	0.1700	0.3300
Precision (Terminated)	0.7800	0.7800	0.7900
Recall (Terminated)	0.9500	0.9400	0.9300
F1-score (Terminated)	0.8600	0.8500	0.8500
Precision (Waiting)	0.3600	0.2800	0.5000
Recall (Waiting)	0.0300	0.4700	0.0100
F1-score (Waiting)	0.0500	0.3500	0.0100
Macro Avg Precision	0.5900	0.6100	0.6200
Macro Avg Recall	0.3900	0.4900	0.4100
Macro Avg F1-score	0.4300	0.4800	0.4400
Weighted Avg Precision	0.7500	0.7500	0.7500
Weighted Avg Recall	0.7700	0.7700	0.7600
Weighted Avg F1-score	0.7300	0.7300	0.7400

(Figure 14: Accuracy Comparison of all Models)

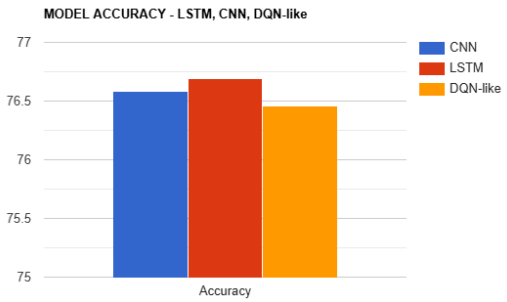
The CNN's performance indicates that there are indeed local patterns in the feature vector that are useful for prediction, but its inability to generalise well to certain minority classes suggests these patterns are not universally applicable. The DQN-like model's performance serves as a useful baseline, demonstrating that a deep but architecturally simple feedforward network is less effective than the specialised CNN and LSTM models for this task. Its high loss and poor minority class performance underscore the value of architectural inductive

biases that align with the data's structure (Peng, Wu and Xiao, 2023). The training curves for all models were stable over the 10 epochs, with validation loss and accuracy tracking the training metrics well. This suggests that the regularisation techniques (Dropout) were effective and that the models were not simply memorising the training data. The main challenge isn't overfitting but severe class imbalance. Models excel on common cases but struggle with rarer, possibly more important, ones. This highlights the need for advanced imbalance handling techniques as a future focus work.

6.6 Benchmarking Results

The findings and results of this Research work was compared against the results reported in previous research on predictive modelling for large scale computing environments.

Bommala et al. (2023) used a machine learning based failure prediction model for cloud tasks, which had resulted in a 70-75% range of accuracy with classical algorithms like, SVM and decision trees. They were less effective in capturing the complex patterns in the high-dimensional system data, even though they offered interpretability and lower computation costs. The models that were developed in my research outperformed these, achieving 76.58% (CNN), 76.69 (LSTM), and 76.46 (DQN-like) accuracy.



(Figure 15: CNN, LSTM, DQN Accuracy comparison)

In Zhao and Wu (2021), a reinforcement learning approach for large scale scheduling with integrating deep neural networks into the scheduling decision process was explored. While their main focus was on action value prediction rather than classification, their results showed that deeper architecture can improve decision making accuracy, just like the findings from my research which proved specialized deep architecture like LSTM outperform the generic feedforward design.

Zhu et al. (2021) proved that, when CNN is adapted to non-image data, it can identify local patterns effectively, but it struggles with minority class detection in the imbalanced datasets. This is similar to my current research work's CNN results, which achieved strong performance on the dominant 'Terminated' class (precision 0.78, recall 0.95), but poor recall for minority states such as 'Running' and 'Waiting'.

Overall, these comparisons prove that:

- Deep learning architectures, particularly LSTM networks, are more capable in handling complex, high-dimensional system task data, than classical ML approaches.
- The class imbalance problem remains a common challenge, as noted in previous studies.

7 Conclusion and Future Work

This research conducted a comparative analysis of three distinct deep learning architectures—CNN, LSTM, and a DQN-like network—for predicting task status in a large-scale computing environment. After navigating a complete research pipeline from raw data processing to rigorous model evaluation, the study found that while all architectures achieved similar overall accuracy scores around 76.5%, these metrics masked significant performance disparities. A deeper analysis of per-class metrics revealed the LSTM model as the most effective. Although its accuracy was only marginally higher at 76.69%, it demonstrated a significantly more balanced predictive profile, distinguished by its superior ability to identify the 'Waiting' minority state. In contrast, the CNN showed competence but struggled with minority classes, and the DQN-like network served as the weakest baseline, highlighting the importance of specialised architectures. These findings suggest that for static feature sets, an architecture like LSTM holds a distinct advantage by modelling complex interdependencies.

Despite these findings, the study is limited by the severe class imbalance in the dataset, which resulted in poor performance on minority classes like 'Running' and 'Waiting'. Future work must address this directly, employing techniques such as class weighting or synthetic oversampling with methods like SMOTE to train more practically useful models. Predictive power could also be enhanced through more comprehensive feature engineering, integrating data from machine metrics and job-level tables to provide richer contextual input. The most promising avenue for future research, however, is the implementation and evaluation of the designed hybrid model. By combining the diverse feature extraction capabilities of CNNs, LSTMs, and dense networks, this model is hypothesised to achieve more robust and accurate predictions across all classes. Therefore, training this hybrid system and comparing its performance against the individual components is the logical and most impactful next step for this research.

References

Ahsan, M.M., Mahmud, M.A.P., Saha, P.K., Gupta, K.D. & Siddique, Z. (2021). Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, 9(3), p.52. **URL:** *https://doi.org/10.3390/technologies9030052*

Akgun, D. (2022). TensorFlow-based deep learning layer for local derivative patterns. *Software Impacts*, p.100452. **URL:** *https://doi.org/10.1016/j.simpa.2022.100452*

Alem, A. & Kumar, S. (2022). Deep learning models performance evaluations for remote sensed image classification. *IEEE Access*, 10, pp.111784–111793. **URL:** *https://doi.org/10.1109/access.2022.3215264*

Alghazzawi, D., Bamasag, O., Albeshri, A., Sana, I., Ullah, H. & Asghar, M.Z. (2022). Efficient prediction of court judgments using an LSTM+CNN neural network model with an optimal feature set. *Mathematics*, 10(5), p.683. **URL:** *https://doi.org/10.3390/math10050683*

Kulshrestha, A., Chang, L. & Stein, A. (2022). Use of LSTM for sinkhole-related anomaly detection and classification of InSAR deformation time series. *IEEE Journal of Selected Topics in Applied Earth*

Observations and Remote Sensing, 15, pp.4559–4570. **URL:** *https://doi.org/10.1109/jstars.2022.3180994*

Awang Pon, M.Z. & Prakash K K, K. (2021). Hyperparameter tuning of deep learning models in Keras. *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing*, 1(1), pp.36–40. **URL:** *https://doi.org/10.55011/staiqc.2021.1104*

Bommala, H., V., U.M., Aluvalu, R. & Mudrakola, S. (2023). Machine learning job failure analysis and prediction model for the cloud environment. *High-Confidence Computing*, 3(4), p.100165. **URL:** *https://doi.org/10.1016/j.hcc.2023.100165*

Deepanshi. (2021). Convolutional neural network with implementation in Python. *Analytics Vidhya*. **URL:** *https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/*

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*. **URL:** *https://doi.org/10.48550/arXiv.2010.11929*

Du, Y., Li, J., Li, C. & Duan, P. (2022). A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, pp.1–15. **URL:** *https://doi.org/10.1109/TNNLS.2022.3208942*

FasterCapital. (2025). Standard deviation: Understanding the three sigma limits of variability. **URL:** *https://fastercapital.com/content/Standard-deviation--Understanding-the-Three-Sigma-Limits-of-Variability.html*

Hinson, J.S., Klein, E., Smith, A., Toerper, M., Dungarani, T., Hager, D., Hill, P., Kelen, G., Niforatos, J.D., Stephens, R.S., Strauss, A.T. & Levin, S. (2022). Multisite implementation of a workflow-integrated machine learning system to optimize COVID-19 hospital admission decisions. *npj Digital Medicine*, 5(1). **URL:** *https://doi.org/10.1038/s41746-022-00646-1*

Jain, P.K., Saravanan, V. & Pamula, R. (2021). A hybrid CNN-LSTM: A deep learning approach for consumer sentiment analysis using qualitative user-generated contents. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 20(5), pp.1–15. **URL:** *https://doi.org/10.1145/3457206*

Kephart, J.O. & Chess, D.M. (2023). The vision of autonomic computing. *Computer*, 36(1), pp.41–50. **URL:** *https://doi.org/10.1109/mc.2003.1160055*

Ketu, S. & Mishra, P.K. (2021). India perspective: CNN-LSTM hybrid deep learning model-based COVID-19 prediction and current status of medical resource availability. *Soft Computing*. **URL:** *https://doi.org/10.1007/s00500-021-06490-x*

Kumar Dubey, A., Kumar, A., García-Díaz, V., Kumar Sharma, A. & Kanhaiya, K. (2021). Study and analysis of SARIMA and LSTM in forecasting time series data. *Sustainable Energy Technologies and Assessments*, 47, p.101474. **URL:** *https://doi.org/10.1016/j.seta.2021.101474*

Li, Y., Wang, Z., Han, R., Shi, S., Li, J., Shang, R., Zheng, H., Zhong, G. & Gu, Y. (2023). Quantum recurrent neural networks for sequential learning. *Neural Networks*, 166, pp.148–161. **URL:** *https://doi.org/10.1016/j.neunet.2023.07.003*

Liang, X., Wu, L., Li, J., Wang, Y., Meng, Q., Qin, T., Chen, W., Zhang, M. & Liu, T.-Y. (2021). R-Drop: Regularized dropout for neural networks. *arXiv preprint arXiv:2106.14448*. **URL:** *https://doi.org/10.48550/arxiv.2106.14448*

Liu, J., Liu, R. & Zhang, X. (2022). Recursive grouping and dynamic resource allocation method for large-scale multi-objective optimization problem. *Applied Soft Computing*, 130, p.109651. **URL:** *https://doi.org/10.1016/j.asoc.2022.109651*

Mumuni, A. & Mumuni, F. (2024). Automated data processing and feature engineering for deep learning and big data applications: a survey. *Journal of Information and Intelligence*, 3(2). **URL:** *https://doi.org/10.1016/j.jiixd.2024.01.002*

Papageorgiou, G., Sarlis, V. & Tjortjis, C. (2024). Unsupervised learning in NBA injury recovery: Advanced data mining to decode recovery durations and economic impacts. *Information*, 15(1), p.61. **URL:** *https://doi.org/10.3390/info15010061*

Peng, H., Wu, C. & Xiao, Y. (2023). CBF-IDS: Addressing class imbalance using CNN-BiLSTM with focal loss in network intrusion detection system. *Applied Sciences*, 13(21), pp.11629–11629. **URL:** *https://doi.org/10.3390/app132111629*

Dhawas, P., Dhore, A., Bhagat, D., Pawar, R.D., Kukade, A. & Kalbande, K. (2023). Big data preprocessing, techniques, integration, transformation, normalisation, cleaning, discretization, and binning. *Advances in Business Information Systems and Analytics*, pp.159–182. **URL:** *https://doi.org/10.4018/979-8-3693-0413-6.ch006*

ProjectPro. (2025). The ultimate guide to building your own LSTM models. [Online] **URL:** *https://www.projectpro.io/article/lstm-model/832*

Rosebrock, A. (2021). Siamese networks with Keras, TensorFlow, and deep learning. [Online] PyImageSearch. **URL:** *https://pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learning/*

Said, O. & Tolba, A. (2021). Accurate performance prediction of IoT communication systems for smart cities: An efficient deep learning-based solution. *Sustainable Cities and Society*, 69, p.102830. **URL:** *https://doi.org/10.1016/j.scs.2021.102830*

Bandal, M.S. (2022). Quantum-inspired heuristic optimization for large-scale cloud resource allocation. *International Journal of Science and Research (IJSR)*, 11(10), pp.1477–1481. **URL:** *https://doi.org/10.21275/sr221011091902*

Scher, S. & Messori, G. (2021). Ensemble methods for neural network-based weather forecasts. *Journal of Advances in Modeling Earth Systems*, 13(2). **URL:** *https://doi.org/10.1029/2020ms002331*

Semmelrock, H., Kopeinik, S., Theiler, D., Ross-Hellauer, T. & Kowald, D. (2023). Reproducibility in machine learning-driven research. *arXiv preprint* arXiv:2307.10320. **URL:** *https://doi.org/10.48550/arxiv.2307.10320*

Shafiq, M. & Gu, Z. (2022). Deep residual learning for image recognition: A survey. *Applied Sciences*, 12(18), p.8972. **URL:** *https://doi.org/10.3390/app12188972*

Shiri, F.M., Perumal, T., Mustapha, N. & Mohamed, R. (2024). A comprehensive overview and comparative analysis on deep learning models. *Journal on Artificial Intelligence*, 6(1), pp.301–360. **URL:** *https://doi.org/10.32604/jai.2024.054314*

Siahaan, D., Fitrianto, A. & Notodiputro, K.A. (2024). Multiple classifier system for handling imbalanced and overlapping datasets on multiclass classification. *ComTech: Computer, Mathematics and Engineering Applications*, 15(1), pp.41–51. **URL:** *https://doi.org/10.21512/comtech.v15i1.11295*

Şimşek, M. & Daş, A.S. (2022). The effect of handling imbalanced datasets methods on prediction of entrepreneurial competency in university students. *Turkish Journal of Forecasting*. **URL:** *https://doi.org/10.34110/forecasting.1185545*

Sony, S., Gamage, S., Sadhu, A. & Samarabandu, J. (2022). Multiclass damage identification in a full-scale bridge using optimally tuned one-dimensional convolutional neural network. *Journal of Computing in Civil Engineering*, 36(2). **URL:** *https://doi.org/10.1061/(asce)cp.1943-5487.0001003*

Tian, J., Liu, Q., Zhang, H. & Wu, D. (2021). Multi-agent deep reinforcement learning based resource allocation for heterogeneous QoS guarantees for vehicular networks. *IEEE Internet of Things Journal*, pp.1–1. **URL:** *https://doi.org/10.1109/jiot.2021.3089823*

DataChannel (2025). What is ETL and how the ETL process works? [Online] **URL:** *https://www.datachannel.co/blogs/what-is-etl-and-how-the-etl-process-works*

Xu, Y., Li, Z., Wang, S., Li, W., Sarkodie-Gyan, T. & Feng, S. (2021). A hybrid deep-learning model for fault diagnosis of rolling bearings. *Measurement*, 169, p.108502. **URL:** *https://doi.org/10.1016/j.measurement.2020.108502*

Zhao, X. & Wu, C. (2021). Large-scale machine learning cluster scheduling via multi-agent graph reinforcement learning. *IEEE Transactions on Network and Service Management*, pp.1–1. **URL:** *https://doi.org/10.1109/tnsm.2021.3139607*

Zhou, K., Oh, S.-K., Qiu, J., Pedrycz, W., Seo, K. & Yoon, J.H. (2024). Design of hierarchical neural networks using deep LSTM and self-organizing dynamical fuzzy-neural network architecture. *IEEE Transactions on Fuzzy Systems*, 32(5), pp.2915–2929. **URL:** *https://doi.org/10.1109/tfuzz.2024.3361856*

Zhu, Y., Brettin, T., Xia, F., Partin, A., Shukla, M., Yoo, H., Evrard, Y.A., Doroshow, J.H. & Stevens, R.L. (2021). Converting tabular data into images for deep learning with convolutional neural networks. *Scientific Reports*, 11(1). **URL:** *https://doi.org/10.1038/s41598-021-90923-y*