Optimizing Demand Forecasting: A Framework With Bayesian Optimization Embedded Reinforcement Learning for Combined Algorithm Selection and Hyperparameter Optimization

Zizhe Wang
wang_zizhe@ihpc.a-star.edu.sg

Xiao Feng Yin
yinxf@ihpc.a-star.edu.sg

Yun Hui Lin
lin_yunhui@ihpc.a-star.edu.sg

Ping Chong Chua
chua_ping_chong@ihpc.a-star.edu.sg

Ning Li
ning_li@ihpc.a-star.edu.sg

Xiuju Fu
fuxj@ihpc.a-star.edu.sg

Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), 1 Fusionopolis Way, #16-16 Connexis, Singapore 138632, Republic of Singapore

***Abstract*—Demand forecasting plays an important role in various fields, and machine learning (ML) has emerged as a prevailing method to perform this task. The selection of an appropriate machine learning algorithm and hyperparameter optimization is essential for accurate predictions. However, this process can be complex and computationally demanding. In this paper, we propose a new framework to effectively tackle the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem. Our method utilizes reinforcement learning (RL) for ML algorithm selection and Bayesian optimization (BO) for hyperparameter tuning. The framework integrates a carefully designed reward function with an ε-greedy policy to guide the system in discovering the best ML pipeline and hyperparameter set. We extensively test the framework on small and large demand forecasting datasets, and the experimental results verify its ability in achieving high forecasting accuracy while significantly reducing computational time.**

Index terms—Demand Forecasting, Bayesian Optimization, Reinforcement Learning, Combined Algorithm Selection and Hyperparameter Optimization

I. Introduction

Demand forecasting plays a pivotal role in various domains, including production planning and supply chain management, as it involves predicting future demand for products or services. Accurate demand forecasts empower organizations to make well-informed decisions, efficiently allocate resources, and maintain optimal inventory levels, leading to enhanced operational efficiency and increased customer satisfaction. While traditional forecasting techniques, such as time series analysis, moving averages methods, and trend analysis, have been widely used, machine learning (ML) has emerged as a prominent and powerful data-driven approach for demand forecasting in recent years. The data-driven nature of ML allows it to capture complex patterns and relationships, thereby enable more accurate predictions compared to traditional methods. However, despite its effectiveness, building a robust machine learning pipeline for demand forecasting still presents several challenges, particularly concerning the optimization of hyperparameters and the selection of the most suitable ML algorithms for complex models or large datasets.

Hyperparameter optimization (HPO) is a critical aspect of ML pipeline construction, as it involves tuning various parameters that significantly impact the model's performance. However, the manual setting and tuning of hyperparameters in machine learning algorithms presents several challenges that require extensive professional expertise and practical experience. Firstly, the relationship between hyperparameter configurations and ML algorithm's performance cannot be explicitly expressed. This lack of transparency renders the hyperparameter optimization akin to a "black box". Secondly, ML algorithms often consist of multiple hyperparameters, each with its own space. As a result, the overall hyperparameter space becomes extensive and highly complex. As hyperparameter optimization typically involves multiple iterations and evaluations, the computational burden increases substantially, making the process resource-intensive and impractical in many cases.

Currently, there are a variety of methods for solving HPO, including basic search methods like grid search [1] and random search [2]. Grid search involves an exhaustive search over an discretized hyperparameter space, evaluating all possible combinations, which can be highly time-consuming. On the other hand, random search randomly selects hyperparameter combinations, saving computation costs but lacks guidance and cannot guarantee optimal results. Another popular approach is the Bayesian optimization (BO) method [3]. Bayesian optimization constructs a probability surrogate model of objective function, which aids in determining the most promising hyperparameter configurations for evaluation in the true objective function. By leveraging the probability model, BO effectively explores the hyperparameter search space and identifies configurations likely to yield better performance. This reduces the reliance on exhaustive evaluations and optimizes the process of finding optimal hyperparameters.

Apart from the HPO problem, selecting an appropriate machine learning (ML) algorithm for demand forecasting can be challenging, given the various available options such as the Random Forest model, XGboost model and Artificial Neural Network (ANN). Each algorithm choice leads to a different ML pipeline, including specific machine learning algorithms and optimized parameter configurations tailored to the unique characteristics of the datasets. The combined optimization of both the ML pipeline selection and corresponding hyperparameter configuration falls within the realm of the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem. Several Automated Machine Learning (AutoML) models have been developed to address this CASH problem, such as Auto-sklearn [4] and Auto-Weka

[5] which belong to the Sequential Model-Based Optimization (SMBO) family. Additionally, Tree-based Genetic Programming (TPOT) [6] employs genetic programming method to design and optimize machine learning pipelines automatically.

Recently, AutoML models based on Reinforcement Learning have garnered significant attention. Hyp-RL [7] applies reinforcement learning in the hyperparameter optimizations. Reinbo [8] combines reinforcement learning with Bayesian optimization for the classification task. However, the application of AutoML models based on Reinforcement Learning to demand forecasting tasks remains relatively scarce. Leveraging these innovative techniques in demand forecasting can potentially improve the efficiency and accuracy of the forecasting process.

In practice, demand forecasting datasets are often large, encompassing several years' worth of sales data for various products. Consequently, HPO for such datasets requires substantial computational efforts. The CASH problem becomes even more complex as each ML algorithm necessitates its own HPO procedure. Despite the above research attempts, there is a lack of research on appropriate and effective AutoML frameworks tailored for demand forecasting tasks. In this paper, we come up with a novel AutoML framework based on Bayesian optimization embedded reinforcement learning. The fundamental idea is to leverage reinforcement learning for ML algorithm selection and Bayesian optimization for hyperparameter optimization. This framework harnesses the strengths of both reinforcement learning and Bayesian optimization, aiming to guide the system in finding the most efficient and accurate ML pipeline with tuned hyperparameters.

The paper is organized into the following sections. Section II briefly introduces the theoretical background of the proposed framework, covering the definition of HPO, a summary of the popular Bayesian optimization methods, and the key components and equations of reinforcement learning. Section III presents the general flow of the proposed framework. The CASH problem is modelled as a two-phase sequential problem, addressing it using reinforcement learning and Bayesian optimization separately. A novel reward function is carefully designed to maximize the reward of the pipeline with high accuracy and efficiency. In Section IV, we carry out several experiments using the proposed framework and discuss the results. Section V gives concluding remarks and summarizes suggestions for future research.

II. Theorical Background

A. HPO problem with Bayesian optimization

Let ζ denote a ML algorithm, with its corresponding hyperparameter space represented by Λ, where $\Lambda = \Lambda_1 \times \Lambda_2 ... \times \Lambda_P$ is the *P*-dimensional hyperparameter space contains both the discrete and continuous values. Each hyperparameter configuration is denoted as λ, where $\lambda \in \Lambda$. Hyperparameter Optimization (HPO) refers to the procedure of finding the optimal hyperparameter set λ^* that yields the best prediction on a given dataset **D** [9] :

$$\lambda^* = \arg\min_{\lambda \in \Lambda} L\left(\zeta\left(\mathbf{D}_{\text{train}}, \lambda\right), \mathbf{D}_{\text{valid}}\right), \tag{1}$$

where *L* denotes the loss function, which serves as objective function during the optimization process. $L\left(\zeta\left(\mathbf{D}_{\text{train}}, \lambda\right), \mathbf{D}_{\text{valid}}\right)$ is the loss calculated by the algorithm ζ with hyperparameter set λ on training set $\mathbf{D}_{\text{train}}$ and evaluated on validation set $\mathbf{D}_{\text{valid}}$.

As mentioned in Section I, Bayesian optimization (BO) methods have emerged as successful approaches for hyperparameter optimization, offering high efficiency and accuracy [3, 10]. The two main components of BO are the probabilistic surrogate model of objective function and the acquisition function to select the subsequential point to evaluate. The prevailing acquisition function in BO is the expected improvement (EI) [10]. Commonly used surrogate models in BO include Gaussian process and tree-based models. Various BO variants have been developed, each with its strengths in specific scenarios. For example, the Sequential Model-Based Algorithm Configuration (SMAC) employs random forest models as surrogate models [11]. Spearmint integrates Gaussian process (GP) modeling and performs well in low-dimensional hyperparameter optimization scenarios. Tree Parzen Estimator (TPE) is renowned for its effectiveness and employs a tree of Parzen estimators for conditional hyperparameters. It excels particularly well in structured HPO tasks [1]. In this paper, we adopt TPE as the chosen BO method for Hyperparameter Optimization.

B. Reinforcement learning

In reinforcement learning (RL), the initial step involves modeling the problem using a Markov Decision Process (MDP). A typical MDP is characterized by several key components: the agent, the environment, states, actions, and rewards. The agent refers to the operational system that we aim to construct and train using reinforcement learning. The environment represents the physical or virtual world in which the agent operates and interacts. The state, denoted as *S*, represents the set of all valid situations in the environment. The action, denoted as *A* refers to the set of choices available to the agent to interact with the environment. The reward, denoted as *R* refers to the positive or negative reinforcement that the agent receives after performing the actions in the environment. In general, the RL algorithms can be categorized as model-based algorithms and model-free algorithms. The model-free algorithm directly learns the optimal policy from interacting with the environment. Model-free algorithms like Q-learning, Deep Q Networks, and SARSA are used when the environment is complex, and its internal dynamics are not known [12]. In this paper, we aim to build a robust framework to solve the CASH problem, thus the model-free algorithm is selected. The Q-learning method is adopted due to its efficiency in updating the RL iteration. Assume the agent's action is controlled by certain policy π, the $Q(s, a)$ represents the state-action value can be expressed as

$$Q_\pi(s,a) = \mathrm{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| S_0 = s, A_0 = a\right], \tag{2}$$

where the $s \in S$, $a \in A$, $r \in R$, γ is a discount factor that balances the significance of immediate and future rewards.

The goal of the RL model is to maximize the cumulative reward by choosing the different actions under the certain policy, written as

$$\pi^*(s) \in \arg\max_a Q^*(s,a) , \tag{3}$$

where $Q^*(s,a)$ is the optimal state-action values.

III. Bayesian Optimization Embedded Reinforcement Learning for CASH

Section II discusses Bayesian optimization, and reinforcement learning. In this paper, we come up with a new framework that combines Bayesian optimization and reinforcement learning to address the challenging CASH problem. Specifically, Bayesian optimization is utilized for Hyperparameter Optimization (HPO), while reinforcement learning is employed for the selection of the ML algorithm in the CASH problem. The detailed formulation of this framework is presented in this section.

A. *Two-phase sequential problem*

In the context of the CASH problem, both the ML pipeline and hyperparameters exist within a conditional hierarchical space. This means that certain hyperparameters are only valid and applicable when specific pipelines are present. In our study, three widely used ML algorithms are studies for pipeline optimization. As depicted in Fig. 1, the scenario involves data pre-processing and feature engineering being selected, representing an incomplete pipeline. To complete the pipeline, one of three ML pipelines needs to be selected, as depicted by the dashed arrow. Once an algorithm is selected, the corresponding hyperparameters associated with that specific algorithm become valid and relevant, as indicated by the solid arrow. This conditional relationship ensures that the hyperparameters are meaningful and applicable within the context of the chosen algorithm, enabling a more tailored and accurate optimization process.

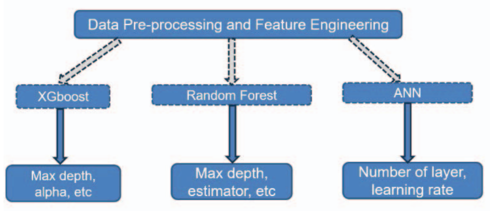
Data Pre-processing and Feature Engineering
XGboost
Random Forest
ANN
Max depth, alpha, etc
Max depth, estimator, etc
Number of layer, learning rate

Fig. 1. Example of ML pipeline selection and hyperparameter tuning

Based on the information presented in Fig. 1, we can conceptualize the pipeline selection and hyperparameter configuration problem as a two-phase sequential problem. In the first phase, a certain algorithm guides the agent to select a path representing an ML pipeline. The second phase is the HPO based on the selected algorithm to give the tunned hyperparameter set. In this phase, the context is determined by the path selected in the first phase.

To model the first phase, the RL algorithm is implemented. The action a_i represents the selection of the machine learning algorithm for demand forecasting. In the second phase, the BO is employed for the HPO on the given hyperparameter space. The two phases are connected by the reward function, which should be designed carefully. The objective of this BO embedded RL model is to find the best algorithm with optimized hyperparameters for the demand forecasting task. The RL process is iterative, with several episodes wherein the algorithm selection action is performed in the first phase, followed by HPO in the second phase. A policy is designed to balance the exploration of different pipelines in the initial episodes and the exploitation of the most promising pipeline in the later episodes. In this model, we use the tabular Q-learning algorithm for RL, which constructs a Q-table to guide the agent's actions. The Q-table is updated iteratively based on the rewards received during the RL process, which will guide the agent's actions in RL.

B. *Initialization of the Q table*

Assuming there are k available ML algorithms for selection in the first phase, the RL undergoes m episodes of training. We can construct a Q-table with m rows and k columns to represent the state-action values. At the beginning, hyperparameters spaces are defined for different ML algorithms, and the Q-values in the Q-table are all set to 0. However, these initial zero values may not provide clear guidance for the agent to make informed decisions.

To address this issue, we take a proactive approach to gather some information about the available ML pipelines. Each of the k available ML pipelines is tested using Bayesian Optimization (BO) for a few steps, typically 3-5 steps. During this BO process, the objective function or loss function is set to the mean absolute percentage error (MAPE), which is evaluated using cross-validation. The MAPE is expressed as

$$L = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{\hat{y}-y}{y}\right|, \tag{4}$$

where n represents the total number of data points, y denotes the actual demand values, and represents the demand values predicted by the ML pipeline [13]. According to Eq. (4), a lower MAPE value indicates a more accurate prediction. A MAPE of 0% indicates a perfect prediction, where the forecast matches the actual values precisely.

The initial Q-value of different pipeline is set as

$$Q(s,a) = 1 - L\,. \tag{5}$$

By using this initial Q-value, the pipeline with lower MAPE will have a larger Q-value. The optimal value for this initial Q-value is 1, corresponding to a MAPE of 0%. A large Q-value in the Q-table indicates the promising pipeline. However, it should be noted that the BO process is only carried out for a few steps, and as a result, the ML pipelines will require further exploration. While the initial Q-values provide a starting point for the RL agent, the pipelines may need more iterations of BO to obtain accurate hyperparameter configurations and improve their performance. After initializing the Q-table, the tuned hyperparameters of different pipelines are stored in the cache for subsequent steps. This caching mechanism allows us to reuse the hyperparameter configurations found during the BO process, which can save computational time and resources in future iterations.

C. *ε-greedy policy*

One crucial aspect of reinforcement learning (RL) is selecting actions that balance exploration and exploitation. In this paper, we adopt the ε-greedy policy to achieve this balance. The ε value is decayed exponentially along the episodes to gradually shift from exploration to exploitation. Let us denote the maximum ε value as ε_{max} (set to 1) and minimum ε value as ε_{min} (set to 0). The decay rate is represented by κ, the ε value at episode m is given

$$\varepsilon(m) = \varepsilon_{\min} + (\varepsilon_{\max} - \varepsilon_{\min})\exp(-\kappa * m)\,. \tag{6}$$

During each episode of RL, a random number is generated using a uniform distribution in the range from 0 to 1. This random number is then compared with the ε value. If the random number is smaller than ε, then the RL agent takes a random action, meaning that it selects an arbitrary pipeline and performs BO based on the cached hyperparameter values stored previously. This approach allows the RL agent to explore various possible pipelines at the initial stages of training when the ε value is relatively high. However, if the random number is larger than ε, the RL agent chooses the pipeline with the largest Q-value. In this case, BO is performed to further tune the hyperparameter set for the selected pipeline. As the RL agent advances through episodes and the ε value decays, it becomes increasingly likely that the RL agent will choose the pipeline with the highest Q-value and exploit the most promising pipeline.

The ε-greedy policy ensures that the RL agent efficiently explores different pipelines in the early stages of training while gradually transitioning to exploit the pipelines with the highest Q-values in later stages. This approach strikes a balance between exploration and exploitation, enabling the RL agent to learn the best ML pipelines and hyperparameters for demand forecasting tasks effectively.

D. *Update of the Q-table and design of reward function*

Once the action is chosen and the BO is carried out to tune the hyperparameter, the Q-table is updated at each episode using the Bellman equation:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{\alpha'} Q(s',a') - Q(s,a)\right], \qquad (7)$$

where α is the learning rate, γ is the discount factor to balance the importance of future reward r in update process. The reward function plays a crucial role in connecting the iterations of continuous episodes in the RL algorithm and should be thoughtfully designed. In this paper, we propose a reward function that considers both the accuracy and efficiency of the ML algorithm. During the initialization stage, the trial time of the BO process along different pipelines at a certain episode is recorded. The mean trial time of BO along each pipeline is then calculated, representing the efficiency of the different pipelines. Once the RL algorithm is running, and BO is performed on the selected pipeline, the new mean trial time is added to the trial time database. These recorded trial time data are then normalized to a scale from 1 to 5 at each episode, denoted as T_{trial}. The objective of the BO-embedded RL algorithm is to choose the best ML pipelines with both high efficiency and good accuracy for demand forecasting. Therefore, the reward function is set as follows:

$$r = (1 - L) / \sqrt{T_{\text{trail}}} \ . \qquad (8)$$

According to the reward function, the pipeline with lower MAPE and lower trial time will receive a larger reward, resulting in a larger update to its Q-value. This design ensures that pipelines with both high efficiency and accuracy receive the largest reinforcement and are more likely to be selected in future episodes.

By incorporating efficiency into the reward function, the RL agent learns to prioritize pipelines that not only yield accurate predictions but also do so efficiently, saving computational time and resources. This helps the RL agent converge to the most suitable ML pipeline with optimized hyperparameters for demand forecasting tasks effectively. The reward function plays a vital role in guiding the RL agent towards the best possible solutions in the CASH problem.

E. *Overall framework*

The previous sections have discussed the key components of the BO embedded RL algorithm, the overall framework of the method is summarized in Fig. 2. In the first step, the Q table with k columns and m rows is created, where k is the number of available ML algorithms for selection, m is the number of the episodes in RL. These episodes can be considered as the budget for the RL agent. Then the hyperparameter sets and accuracy results of different ML algorithms obtained from the BO process are stored in a cache. Additionally, the trial time of BO on different pipelines is recorded for evaluating efficiency. Thereafter, the RL agent implements the ε-greedy strategy policy to select actions. Based on the current ε value and a random number, the agent chooses either a random action (representing exploration) or the action with the highest Q-value (representing exploitation). The chosen action corresponds to the selection of a pipeline, and the RL agent performs BO based on the hyperparameter set stored in the cache. In this stage, the cache is updated with more accurate results and the corresponding hyperparameter sets. The trial time database is also updated for the evaluation of the reward using Eq. (8). After completing each episode, the Q-table is updated using Eq. (7). The RL agent continues the iterations until it reaches the budget of episodes (m). This iterative process allows the agent to refine its decisions over time based on the rewards received and the exploration-exploitation trade-off. By the end of the RL iterations, the best ML pipeline and its corresponding optimized hyperparameter set can be found in the cache.

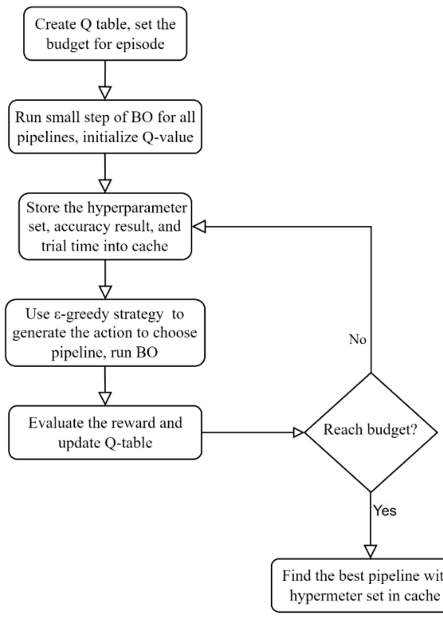
Create Q table, set the budget for episode
Run small step of BO for all pipelines, initialize Q-value
Store the hyperparameter set, accuracy result, and trial time into cache
Use ε-greedy strategy to generate the action to choose pipeline, run BO
Evaluate the reward and update Q-table
Reach budget?
No
Yes
Find the best pipeline with hypermeter set in cache

Fig. 2. Flowchart of the proposed framework

IV. Experiments

A. *Experimental test on the small dataset*

In order to validate the proposed framework, experiments are carried out using a small dataset and subsequently a larger dataset for further testing. For these experiments, three machine learning algorithms were chosen, namely XGBoost, Random Forest, and Artificial Neural Network (ANN). The typical hyperparameter spaces for each of these methods are provided in Table I.

The Triazines dataset, obtained from OpenML, consisting of 186 samples and 60 features, was chosen to thoroughly test and validate the proposed framework. The RL algorithm was executed for 500 episodes, with a learning rate of 0.5 and a discount factor of 0.95 to update the Q-table during training. For the ε-greedy policy, an exponential decay rate of 0.006 was set to balance exploration and exploitation. To evaluate the accuracy of each pipeline, the 1-MAPE metric was employed, where a value close to 1 indicates optimal accuracy in demand forecasting. The objective of the RL algorithm is to find the best ML pipeline with hyperparameter settings that minimize the MAPE.

TABLE I HYPERPARAMETER SPACE FOR ML ALGORITHMS

Operation	Parameter	Range
XGboost	eta	(0.01, 0.2)
XGboost	gamma	(0, 9)
XGboost	max_depth	(3, 20)
XGboost	min_child_weight	(1, 10)
XGboost	subsample	(0.5, 1)
XGboost	reg_lambda	(0.5, 1)
XGboost	colsample_bytree	(0.5, 1)
XGboost	n_estimators	(50, 500)
Random Forest	max_depth	(10, 100)
Random Forest	max_features	['sqrt', 'log2', None]
Random Forest	min_samples_leaf	(2, 20)
Random Forest	min_samples_split	(2, 20)
Random Forest	n_estimators	(50, 1000)
Random Forest	bootstrap	[True, False]
ANN	num_layers	(2, 3)
ANN	units	(16, 1024)
ANN	dropout	(0.25, 0.75)
ANN	Batch_size	(8, 128)
ANN	nb_epochs	(20, 200)
ANN	optimizer	['adadelta', 'adam', 'rmsprop']
ANN	activation	['relu', 'elu', 'selu', 'sigmoid', 'softplus', 'softsign', 'tanh']

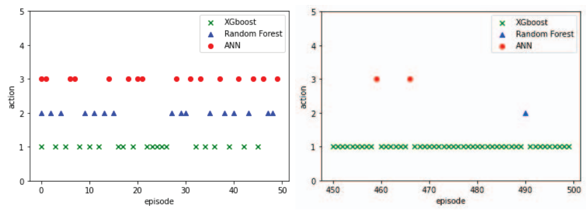

Fig. 3. Actions of first 50 episodes (a) and last 50 episodes (b)

The RL actions during the first 50 episodes and the last 50 episodes were plotted in Fig. 3. In Fig. 3 (a), it can be observed that the RL actions during the initial episodes were nearly random, as all three ML pipelines (XGBoost, Random Forest, and ANN) were tested. This demonstrates that the ε-greedy policy effectively promotes exploration of all possible pipelines in the initial stages of training, ensuring that the RL agent explores different options before focusing on exploiting the most promising ones. Fig. 3 (b), on the other hand, shows that during the last 50 episodes, the RL agent consistently chose the XGBoost algorithm as the most promising pipeline. This choice indicates that the ε-greedy policy gradually shifted the focus to exploitation, as the RL agent learned that the XGBoost pipeline yielded the best results in terms of accuracy and efficiency on the Triazines dataset.

Fig. 4 presents the accuracy results of each pipeline during the training process. In the first episode, the Random Forest pipeline demonstrated the highest accuracy among the three pipelines. However, the ε-greedy policy, wisely incorporated into the RL algorithm, guided the agent to explore all available pipelines in search of potential improvements in accuracy. As a result of this exploration, all three pipelines showed improvements in accuracy. This showcases the effectiveness of the RL agent in efficiently exploring the ML pipeline space, even though Random Forest initially performed well. In the later stages of training, the RL agent consistently selected the XGBoost algorithm due to its high accuracy and low trial time. BO was subsequently applied to further fine-tune the hyperparameters of the XGBoost pipeline, leading to additional improvements in accuracy. However, as the accuracy results of the XGBoost pipeline show, the improvements in accuracy become more challenging to achieve in the later stages of training. This suggests that achieving higher levels of accuracy requires more steps of BO, indicating the diminishing returns in accuracy improvement as the pipeline becomes more optimized.

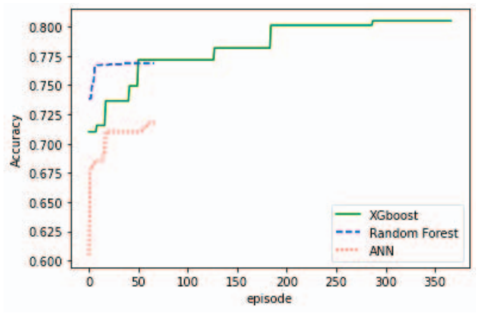

Fig. 4. Accuracy of each pipeline along the episodes

TABLE II SUMMARY OF THE RESULTS OF DIFFERENT PIPELINES

Pipeline	Mean trail time (s)	Number of episodes	Accuracy results
XGboost	1.9284	367	0.8050
Random Forest	4.7139	67	0.7687
ANN	3.5588	68	0.7178

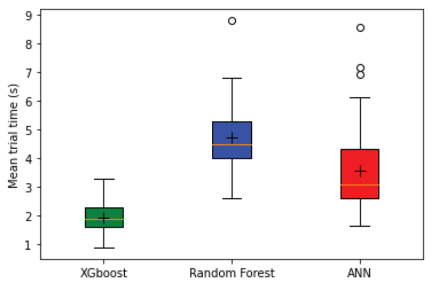

Fig. 5. Boxplot of the trail time of three algorithms

Table II provides a summary of the results evaluated by different ML pipelines during the experimentation phase. From the results in Table II, it is evident that the XGBoost pipeline was selected for the majority of the episodes. This selection was in line with the RL agent's exploration-exploitation strategy, as XGBoost consistently demonstrated the highest accuracy results and the lowest mean trial time among the three pipelines. As mentioned earlier, the trial times of different pipelines at each episode were recorded in the cache for evaluating the reward at different stages. The experiments were conducted on a personal computer with an Intel Core i7-7600U CPU and 12GB RAM. To visualize the trial time distribution of the three algorithms, a boxplot is presented in Fig. 5. The boxplot shows that the mean trial time of XGBoost is the lowest, indicating that it is the most efficient pipeline among the three. Additionally, the boxplot illustrates that XGBoost's trial times are relatively stable with fewer outliers, while both Random Forest and ANN exhibit some outliers with longer trial times.

B. Experimental test on the large dataset

In this experimental test, the orange juice (OJ) dataset from Azure Open Datasets was used. The dataset consists of weekly sales of orange juice in 64-ounce containers for 83 stores in the Chicago area. It contains sales data for 121 weeks and three different brands, resulting in an original dataset with 28947 rows and 17 columns. Before applying the proposed framework, data pre-processing was carried out, which involved data cleaning, feature expansion and selection, and categorical feature encoding. The feature expansion was performed by taking the logarithm of the numerical features and then creating polynomial features of order 2. Furthermore, feature selection was conducted to remove highly correlated features. The processed data resulted in a dataset with 28947 rows and 151 columns.

TABLE III SUMMARY OF THE RESULTS OF DIFFERENT PIPELINES FOR OJ DATASET

Pipeline	**Mean trail time (s)**	**Number of episodes**	**Accuracy results**
XGboost	170.97	15	0.9713
Random Forest	145.42	3	0.9624
ANN	1034.24	4	0.9231

The proposed framework was then implemented on this processed dataset. Given the large size of the dataset, the episode was set to 20, and the decay rate of ε in the ε-greedy policy was set to 0.1. Table III summarizes the results of different pipelines for the OJ dataset, indicating that XGboost achieved the best accuracy results.

Notably, for a large dataset like the OJ dataset, the trial time of Bayesian optimization (BO) can be considerable and varies significantly for different pipelines. The proposed framework effectively addresses this issue and saves computational time vastly. For instance, the number of BO trials in this experiment is set as 3, if the dataset is tested on the ANN algorithm with 20 episodes, the total running time can be estimated as $1034.24 \times 3 \times 20 = 17.23$ hours. However, the proposed framework avoids wasting computational resources on ANN pipeline, the total running time is $170.97 \times 3 \times 15 + 145.42 \times 3 \times 3 + 1034.24 \times 3 \times 4 = 5.95$ hours, the accuracy result is also better. Overall, the experimental results on the OJ dataset validate the efficiency and accuracy of the proposed Bayesian optimization embedded reinforcement learning framework in selecting the optimal ML pipeline and hyperparameters for demand forecasting, especially when dealing with large datasets with varying trial times for different pipelines.

V. CONCLUSIONS AND FUTURE WORK

This paper introduces a novel AutoML framework designed to tackle the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem for demand forecasting tasks. By utilizing reinforcement learning for machine learning algorithm selection and Bayesian optimization for hyperparameter optimization, the framework aims to find the most effective ML pipeline with optimized hyperparameters. The proposed reward function maximizes pipeline accuracy while ensuring computational efficiency.

The ε-greedy policy employed in the framework enables exploration of various pipelines at the beginning and exploitation of the most promising ones as the process progresses. Experimental tests on both small and large datasets validate the effectiveness of the proposed framework in identifying optimal ML pipelines and corresponding hyperparameters. Additionally, the framework significantly reduces computational effort by avoiding wasteful computations on unpromising pipelines, especially for large datasets.

For future work, it would be interesting to expand the case study by incorporating more ML pipelines to assess the final accuracy results and computational resource allocation. The different feature engineering techniques can also be considered in different ML pipelines. Furthermore, enhancing the framework with a mechanism to automatically determine the completion of algorithm selection and hyperparameter optimization could further improve its efficiency and adaptability to different scenarios.

ACKNOWLEDGMENT

This research is supported by A*STAR under Supply Chain (SC) 4.0 – Digital Supply Chain Development via Platform Technologies Programme (Grant No. M21J6a0080). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of A*STAR.

REFERENCES

[1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems,* vol. 24, 2011.

[2] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research,* vol. 13, no. 2, 2012.

[3] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, 2011: Springer, pp. 507-523.

[4] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," *Advances in neural information processing systems,* vol. 28, 2015.

[5] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms," *CoRR, abs/1208.3719,* 2012.

[6] R. S. Olson and J. H. Moore, "TPOT: A tree-based pipeline optimization tool for automating machine learning," in *Workshop on automatic machine learning*, 2016: PMLR, pp. 66-74.

[7] H. S. Jomaa, J. Grabocka, and L. Schmidt-Thieme, "Hyp-rl: Hyperparameter optimization by reinforcement learning," *arXiv preprint arXiv:1906.11527,* 2019.

[8] X. Sun, J. Lin, and B. Bischl, "Reinbo: Machine learning pipeline search and configuration with bayesian optimization embedded reinforcement learning," *arXiv preprint arXiv:1904.05381,* 2019.

[9] M. Feurer and F. Hutter, "Hyperparameter optimization," *Automated machine learning: Methods, systems, challenges,* pp. 3-33, 2019.

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems,* vol. 25, 2012.

[11] M. Lindauer and F. Hutter, "Warmstarting of model-based algorithm configuration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, vol. 32, no. 1.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[13] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi, "Mean absolute percentage error for regression models," *Neurocomputing,* vol. 192, pp. 38-48, 2016.