

Article

An Online Drift-Adaptive Framework for Semantic Representation and Classification of HPC Jobs

Xiaotao Xi , Gongju Guo * and Jianxiang Gu 

Key Laboratory of Spatial-Temporal Big Data Analysis and Application of Natural Resources in Megacities (Ministry of Natural Resources), Shanghai Municipal Institute of Surveying and Mapping, No. 419 Wuning Road, Shanghai 200063, China; xtxi@shsmi.cn (X.X.); gujx@shsmi.cn (J.G.)

* Correspondence: guoda@shsmi.cn

Abstract

With the rapid advancement of computing technologies, high-performance computing (HPC) systems have continued to expand in scale and have been widely applied across various domains. To improve resource utilization, existing studies commonly employ offline-trained models based on historical resource usage data to classify HPC jobs as compute-bound or memory-bound. However, such methods typically rely on static historical data and struggle to adapt to dynamically changing workloads over time, resulting in degraded prediction performance and reduced stability under concept drift. To address this issue, this paper proposes an online drift-adaptive framework for HPC job semantic representation and classification (DA-HJSC). The framework employs a hybrid mechanism that combines offline semantic learning with online dynamic adaptation. Specifically, an offline classification model is first trained using semantic representations of job descriptions together with user historical behavioral features (UHBF). During the inference stage, an exponentially weighted moving average (EWMA) mechanism is further introduced to dynamically fuse the output probabilities of the classification model, thereby producing the final classification results for HPC jobs. Experimental results on the publicly available F-DATA dataset covering the period from 1 May 2023 to 30 April 2024 demonstrate that DA-HJSC consistently improves HPC job classification performance across different pre-trained language models and base classifiers, achieving a maximum $F1_{macro}$ (overall) score of 0.9272. Compared with the corresponding baselines, DA-HJSC reduces the number of low- and medium-performance days by up to 11 and 35 days, respectively, across all experimental configurations. Overall, the DA-HJSC framework achieves an effective balance among classification accuracy, performance stability, and online adaptability, providing a solution with practical deployment potential for dynamic HPC workload classification.



Academic Editors: Xiaodong Yu,
Shaoyi Huang and Zhen Xie

Received: 4 June 2026

Revised: 30 July 2026

Accepted: 31 July 2026

Published: 2 August 2026

Copyright: © 2026 by the authors.
Licensee MDPI, Basel, Switzerland.
This article is an open access article
distributed under the terms and
conditions of the [Creative Commons
Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Keywords: high-performance computing; concept drift; job classification; semantic representation; online drift adaptation

1. Introduction

With the rapid development of artificial intelligence (AI) technologies across various industries, high-performance computing (HPC) clusters, with their powerful parallel computing capabilities, have become essential infrastructure for supporting AI model training, large-scale data analysis, and scientific computing [1–4]. In practical applications, a large number of users submit different types of jobs to HPC clusters, and these jobs exhibit significant differences in computational patterns, resource requirements, and runtime

characteristics. For example, some jobs are compute-bound and require substantial CPU or GPU computing resources, while others are memory-bound and rely more heavily on memory bandwidth and data access efficiency. Without effective job identification and resource management mechanisms, resource allocation may become inefficient, thereby reducing the overall system utilization. Therefore, accurately identifying the characteristics of user-submitted jobs to support appropriate and efficient resource scheduling has become a key issue in HPC system management and scheduling optimization [5].

To improve resource utilization in HPC clusters, a common approach is to collect resource utilization data and runtime performance metrics using monitoring tools [6] and then apply empirical rules or machine learning models to analyze job types and resource requirements based on historical runtime data [7], thereby facilitating coordinated scheduling of computing and storage resources. These methods can identify workload patterns in historical data and improve resource scheduling efficiency to a certain extent. However, such methods typically rely on offline analysis and therefore struggle to respond promptly to evolving workload demands. Over time, users' job submission patterns may change due to factors such as algorithm updates, variations in data scale, or adjustments in task types. These changes can alter the feature distributions of submitted jobs, resulting in concept drift [8]. When a model cannot adapt to such changes in a timely manner, its generalization ability and prediction performance gradually degrade, which in turn affects the accuracy of resource scheduling decisions and increases system operational overhead and management complexity. Therefore, developing a lightweight HPC job classification method that is suitable for online deployment and can adapt to dynamic changes in users' job submission behavior while maintaining prediction accuracy and reducing model complexity remains an important technical challenge in HPC job classification and resource scheduling.

To address these issues, this paper proposes an online drift-adaptive framework (DA-HJSC) for semantic representation and classification of HPC jobs. The framework integrates semantic information of user-submitted jobs, user historical behavioral features (UHBF), and a probability fusion based on exponentially weighted moving average (EWMA), thereby enabling dynamic classification of HPC jobs. The innovation of the framework lies in modeling users' long-term behavioral patterns during offline training and adapting to short-term changes in class distributions during online inference, thereby enabling online adaptation of prediction results. Experimental results show that the proposed DA-HJSC framework can effectively adapt to changes in HPC workloads, thus mitigating the impact of concept drift on job classification performance.

The main contributions of this paper are summarized as follows:

- (1) Semantic representations of job descriptions are integrated with UHBF to construct comprehensive feature vectors as inputs to the machine learning model, thereby enhancing the model's capability to capture users' long-term behavioral patterns.
- (2) An online drift-adaptive mechanism based on an EWMA is designed. For each user in an HPC cluster, the mechanism uses the true labels of that user's recently completed jobs to dynamically update that user's job-class distribution, thereby continuously adapting to changes in user behavior and effectively mitigating model performance degradation induced by concept drift.
- (3) The DA-HJSC framework integrates an offline classification model with an online drift-adaptation mechanism to jointly model users' long-term behavioral patterns and short-term dynamic changes. To evaluate the effectiveness and applicability of the framework, this study conducts experiments using three pre-trained language models (PLMs)—namely, all-MiniLM-L6-v2, bge-base-en-v1.5, and e5-base-v2—in combination with two base classifiers, XGBoost and RF, and further compares the

results with those of the ARF method. The experimental results demonstrate that DA-HJSC improves both HPC job classification performance and daily performance stability across different semantic encoding methods and base classifiers while incurring minimal online computational and additional memory overhead. These results indicate that the framework has the potential for practical deployment.

The remainder of this paper is organized as follows. Section 2 introduces related work on HPC workload analysis and concept drift. Section 3 presents the proposed DA-HJSC framework. Section 4 presents the experimental results and corresponding analyses based on a real-world dataset. Section 5 discusses the performance, drift-adaptive capability, and stability of the DA-HJSC framework. Finally, Section 6 concludes the paper and discusses directions for future research.

2. Related Work

2.1. HPC Workload Analysis

In recent years, studies on workload analysis in HPC clusters have mainly focused on several aspects. The first concerns computational performance analysis in HPC clusters [9]. The commonly used Roofline model [10] evaluates computational bottlenecks in hardware and software systems through visualization, supporting resource scheduling decisions [11,12] and hardware-software co-design [13–17]. Diogo Marques et al. [18] proposed the Cache-Aware Roofline Model (CARM), which can analyze the parallel processing limits of multicore processors, optimize the development process, and explore the potential of hardware architectures. Wu et al. [19] developed PRoof, a multifunctional deep neural network performance analysis framework based on the Roofline model, which improves hardware performance while optimizing model design, enabling deep learning models to better align with hardware architectures in HPC cluster environments. These studies characterize the computational characteristics of applications based on the relationship between operational intensity and memory bandwidth, distinguish whether applications are compute-bound or memory-bound, and provide a basis for HPC workload analysis, performance optimization, and resource scheduling.

The second aspect involves using HPC operational data and applying deep learning or machine learning methods to predict and analyze job runtime [20,21] and energy consumption [22,23], as well as to perform fault classification [24,25] in HPC systems. Such studies help improve resource scheduling efficiency and reduce system overhead. For example, Tian et al. [26] proposed a deep learning-based framework that combines traditional offline models with an online incremental update mechanism to predict the power consumption of HPC jobs, thereby improving overall system efficiency. Yokelson et al. [27] adopted a two-stage machine learning approach that combines key feature selection with performance metric prediction to model HPC application performance, achieving relatively accurate predictions of job execution time.

With the development of natural language processing (NLP) technologies, large language models (LLMs) have shown considerable potential for HPC performance analysis and decision-making in complex engineering systems. For example, Bolet et al. [28] explored the use of LLMs to predict whether GPU kernels are compute-bound or memory-bound, and suggested that this approach could serve as a hardware-independent alternative for HPC performance analysis. Zou et al. [29] further explored the semantically guided decision-making capabilities of LLMs in complex engineering systems, providing useful insights into semantic understanding, collaborative decision-making, and resource coordination in complex computing systems. Antici et al. utilized a PLM to extract semantic features from HPC job descriptions and used the generated semantic representations as inputs to machine learning models for downstream tasks, including predicting job power

consumption and runtime [30,31], as well as classifying HPC jobs offline as compute-bound or memory-bound [32,33]. This approach, which combines NLP-based semantic representations with machine learning models, can further enhance the semantic representation of HPC workloads and effectively support resource optimization and intelligent scheduling in HPC systems.

2.2. Concept Drift

In HPC clusters, the distribution of workload performance data continuously changes due to variations in user-submitted job types and runtime behaviors, which may lead to concept drift in classification tasks. Concept drift refers to changes in the statistical properties or underlying distributions of data over time [34,35]. Based on different patterns of change in data distributions, concept drift can generally be categorized into four types: sudden drift, gradual drift, incremental drift, and recurring concepts [36,37]. In recent years, concept drift has become an important research topic in the fields of data analysis and machine learning [38,39]. To address the degradation of model predictive performance caused by concept drift, existing studies mainly focus on two aspects: drift detection [40] and drift adaptation.

In the field of drift detection, Şenol et al. [41] proposed a domain knowledge-enhanced dual-LLM framework that integrates structured cues with pre-trained models for concept drift detection and fraudulent content classification. Ghahramanian et al. [42] proposed an Adaptive Neural Ensemble Network (AdaNEN) for dynamic text-stream classification, which can effectively detect and handle concept drift while maintaining stable predictive performance. Li et al. [43] proposed a Variational Autoencoder for Anomalous Sequences (VAE4AS) method that combines incremental learning with a dual drift-detection mechanism, demonstrating strong performance in handling anomalous sequences in non-stationary streaming data. Greco et al. [44] proposed an unsupervised deep learning framework named DriftLens for real-time concept drift detection and characterization in unstructured data. The framework can also analyze the impact of drift on label distributions. Ali et al. [45] proposed a semi-supervised Autoencoder-based Drift Detection Method (AEDDM), which improves the accuracy of both drift detection and binary classification through a dual-threshold reconstruction error mechanism.

In the field of drift adaptation, Iman et al. [46] designed an adaptive memory-based continual learning method that incorporates domain knowledge and dynamically adjusts model weights to adapt to changing data patterns and alleviate catastrophic forgetting. Pérez et al. [47] proposed a self-training-enhanced semi-supervised learning method for concept drift scenarios that improves classification performance by generating additional labels and optimizing the model learning and adaptation process. Desale et al. [48] proposed a hybrid method that combines an optimized deep convolutional neural network (Deep CNN) with an Optimized Adaptive Sliding Window (OASW) to address concept drift in dynamic data streams. Additionally, complex time-series modeling [49] techniques have been applied to dynamic and non-stationary data, providing useful insights for drift adaptation. Suryawanshi et al. [50] proposed a self-supervised dual-task framework that combines representation learning with a ternary-adaptive ensemble to address concept drift and catastrophic forgetting in non-stationary data streams.

In recent years, concept drift detection and adaptation methods have increasingly been applied to cloud computing environments. Kidane et al. [51] proposed a workload prediction method for cloud computing environments that constructs a knowledge base using historical statistical features and performs automatic parameter tuning and adaptation, thereby improving prediction accuracy while detecting concept drift. Chen et al. [52] proposed a frequent pattern mining algorithm with a two-layer embedded variable sliding

window, which dynamically adjusts the window size to distinguish among different types of concept drift and adapt to them while maintaining high stability and robustness.

However, existing research on concept drift has mainly focused on general streaming data or cloud computing scenarios, while studies on HPC job classification, particularly the classification of compute-bound and memory-bound jobs, remain limited. In addition, continuously updating classification models typically incurs additional training and model maintenance overhead. In light of these limitations, this paper first trains a binary classification model in which the job class labels are derived from HPC performance metrics and the input features are constructed from semantic representations of user-submitted jobs and UHBF. Furthermore, an online adaptive method is proposed to predict whether submitted jobs are compute-bound or memory-bound, thereby improving prediction stability and long-term adaptability in dynamic HPC workload environments characterized by evolving user behavior patterns.

3. Proposed Framework

This section presents a detailed description of the proposed DA-HJSC framework for semantic representation and classification of HPC jobs. The framework consists of two main components: model training and model inference. Before model training, the raw data need to be preprocessed (Section 3.1). In the model training stage, a PLM is first employed to encode user-submitted job descriptions into semantic representations (Section 3.2). These semantic representations are then combined with UHBF (Section 3.3) to construct a comprehensive feature vector, which serves as input to an XGBoost classification model. The ground-truth labels, derived from performance metrics (Section 3.4), are used for supervised training. The model inference stage consists of two parts. The first part updates the EWMA probability estimates for different users or job types in the HPC environment using the ground-truth labels of recently completed jobs (Section 3.5). The second part performs a weighted fusion of the prediction probability produced by the XGBoost model and the short-term probability generated by EWMA (Section 3.6), yielding the final prediction probability. Based on this probability, the submitted job is classified as either compute-bound or memory-bound. Figure 1 illustrates the overall architecture of the proposed drift-adaptive framework.

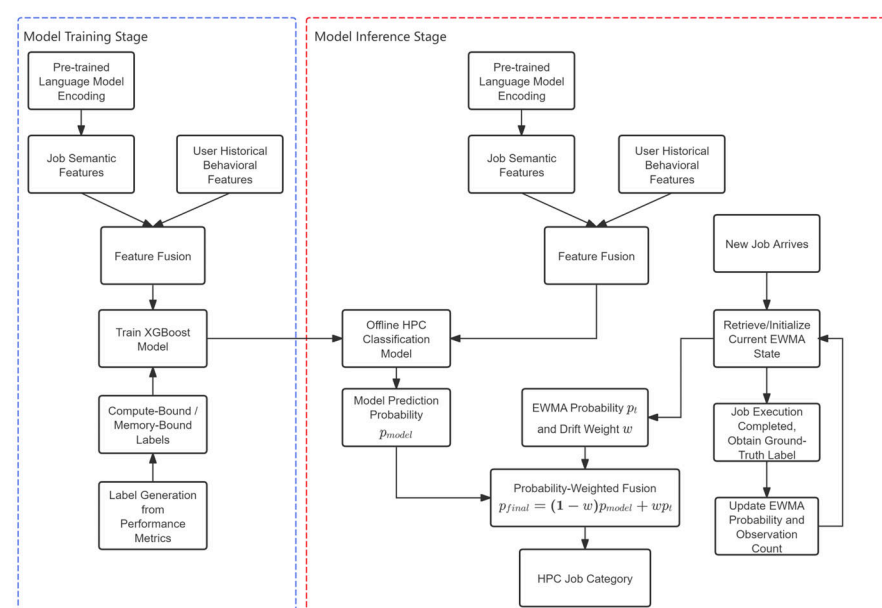


Figure 1. Training and Inference Workflow of the DA-HJSC Framework.

3.1. Data Preprocessing

The HPC cluster runtime dataset used in this study consists of two components: performance metrics and user-submitted job feature data. The performance metrics include job submission time, start time, end time, the number of allocated nodes, memory bandwidth, and FLOPs. The user-submitted job feature data include the username, task name, and the hardware resource requirements specified at submission time, such as the number of cores, frequency, number of nodes, runtime environment, and memory configuration. In the data preprocessing stage, the raw data are first cleaned by filtering records with empty strings in specified fields, converting performance-related fields into numeric data types, and removing duplicate records. Subsequently, the job logs are sorted by timestamp and grouped by job ID (*jid*). For each job, the timestamps generated at the submission, start, and completion stages are aggregated so that each group corresponds to a complete job instance. Based on this process, job-level performance metrics and user-submitted job log data are generated and stored in Parquet format on a monthly basis. Finally, each job instance is uniquely identified by its *jid*, while jobs with the same task name may correspond to multiple distinct job instances.

3.2. Job Feature Encoding

A PLM is employed to encode each user-submitted job description into a semantic representation. Specifically, job-related information, such as the username, task name, and selected resource configuration attributes (e.g., number of CPU cores, CPU frequency, number of nodes, and runtime environment), is organized into a unified textual format and then fed into the PLM to obtain a semantic representation of the job. Through this process, the numerical features and textual information are jointly mapped into a shared semantic representation space, enabling the model to capture job attributes, resource demand characteristics, and potential computational patterns. This unified representation provides discriminative features for downstream classification, thereby helping improve classification accuracy and robustness, particularly for complex job scenarios.

3.3. User Historical Behavioral Features (UHBF)

Historical records of completed jobs submitted by users are aggregated by completion date to construct daily job templates. From multiple monthly job-level data files, relevant fields are extracted, including username, task name, date, job submission time, completion time, and the counts of compute-bound and memory-bound jobs. The time fields are then parsed and normalized, and a completion-day identifier is generated based on the job completion time. On this basis, the job records are grouped by username, task name, and completion day and aggregated to obtain statistics for each group, including the total number of jobs completed on that day, the number of compute-bound jobs, the earliest submission time, the earliest completion time, and the latest completion time. Finally, these statistics are stored in an NPZ file for daily behavioral modeling.

The proposed UHBF construction algorithm is presented in Algorithm 1. Taking the submission time of the current job as a reference, a rolling window is constructed from the historical data of the same user. For a job submitted on day d , UHBF is constructed using only the jobs completed from day $d - N$ through day $d - 1$, excluding all jobs from day d . This ensures that UHBF is derived entirely from historical data preceding the prediction day, thereby preventing temporal information leakage during downstream prediction. When fewer than N days of historical data are available, all available records of jobs completed before day d are used, again excluding any data from day d . The UHBF vector is then computed from the filtered historical data and comprises the proportion of compute-bound jobs, the task-level daily average purity, and the stability of user submission behavior.

Among these features, the proportion of compute-bound jobs represents the share of compute-bound jobs in the historical records and is calculated as follows:

$$User_{p1} = \frac{\sum c_{1k}}{\sum c_k} \quad (1)$$

The task-level daily average purity represents the average majority-class proportion of a user's task-level jobs across historical days, and is calculated as follows:

$$\begin{aligned} purity_k &= \max\left(\frac{c_{1k}}{c_k}, 1 - \frac{c_{1k}}{c_k}\right) \\ avg_purity &= \text{mean}(purity_k) \end{aligned} \quad (2)$$

The stability of user submission behavior is measured by the proportion of stable days among all historical days and the proportion of jobs completed on stable days among all historical jobs, and is calculated as follows:

$$\begin{aligned} stable_k &= \begin{cases} 1, c_k \geq \theta \text{ and } purity_k \geq \eta \\ 0, \text{ otherwise} \end{cases} \\ stable_days_ratio &= \text{mean}(stable_k) \\ stable_jobs_ratio &= \frac{\sum c_k \cdot stable_k}{\sum c_k} \end{aligned} \quad (3)$$

where k denotes an available historical job-completion day within the selected historical window, c_k represents the total number of jobs completed on day k , and c_{1k} denotes the number of compute-bound jobs completed on day k . The term $purity_k$ represents the purity of the job category distribution on day k , which measures the degree to which jobs are concentrated in the majority class on that day. The avg_purity represents the average class purity across all available historical days within the window. The term $stable_k$ indicates whether the job category distribution on day k satisfies a predefined stability condition. The parameters θ and η provide an operational definition of a stable historical day, requiring that the number of jobs be no less than θ and the class purity be no lower than η . In this study, $\theta = 8$ and $\eta = 0.9$. The $stable_days_ratio$ represents the proportion of available historical days within the window that satisfy the predefined stability condition. The $stable_jobs_ratio$ represents the proportion of jobs completed on stable days among all jobs within the historical window.

Algorithm 1: User Historical Behavioral Features (UHBF)

Input: User of the current job u_i , submission time t_i , daily job template T , window size N

Output: UHBF vector f_i of the current job

1. $d_i \leftarrow \text{date}(t_i)$ // extract the submission day from t_i
 2. Retrieve historical records of user u_i from T
 3. Select records whose completion day k satisfies
 $k \in [d_i - N, d_i - 1]$
// Records completed on day d_i are excluded
 4. If no valid records exist, return $f_i \leftarrow [0, 0, 0, 0]$
 5. Within the selected historical window N :
 $c \leftarrow \sum c_k$ //total number of jobs in the window
 $c_1 \leftarrow \sum c_{1k}$ //total number of compute-bound jobs in the window
 6. For each historical day k :
 $p_{1k} \leftarrow c_{1k}/c_k$
 $purity_k \leftarrow \max(p_{1k}, 1 - p_{1k})$
 $stable_k \leftarrow 1, \text{ if } (c_k \geq \theta \text{ and } purity_k \geq \eta); \text{ else } 0$
-

Algorithm 1: *Cont.*

7. Compute features :

$$\begin{aligned} f_1 &\leftarrow c_1/c \quad // \text{ User}_{p1} \\ f_2 &\leftarrow \text{mean}(\text{purity}_k) \quad // \text{ avg_purity} \\ f_3 &\leftarrow \text{mean}(\text{stable}_k) \quad // \text{ stable_days_ratio} \\ f_4 &\leftarrow (\sum c_k \cdot \text{stable}_k)/c \quad // \text{ stable_jobs_ratio} \end{aligned}$$

8. Return $f_i \leftarrow [f_1, f_2, f_3, f_4]$

3.4. Label Generation

Based on the Roofline model, jobs are classified as compute-bound or memory-bound by comparing their operational intensity with the critical operational intensity of the system. The critical operational intensity OI is calculated using the system peak performance P_{peak} and peak memory bandwidth MB_{peak} . The corresponding formula is as follows:

$$OI = \frac{P_{peak}}{MB_{peak}} \quad (4)$$

For a job j , its computational performance, memory bandwidth, and operational intensity are defined as:

$$\begin{aligned} P_j &= \frac{FLOPs_j}{\text{duration}_j \cdot \text{nodes_alloc}_j} \\ MB_j &= \frac{\text{moved_memory_bytes}_j}{\text{duration}_j \cdot \text{nodes_alloc}_j} \\ OI_j &= \frac{P_j}{MB_j} \end{aligned} \quad (5)$$

where $FLOPs_j$ denotes the number of floating-point operations of job j , duration_j denotes the execution time, nodes_alloc_j denotes the number of allocated nodes, and $\text{moved_memory_bytes}_j$ denotes the amount of data moved through memory. These metrics are obtained from job execution statistics and performance monitoring data. If $OI_j \geq OI$, the job is labeled as compute-bound; otherwise, it is labeled as memory-bound.

For Roofline-based label generation, FLOPs and memory traffic are first measured using node-level performance monitoring data and aggregated over the job execution period. For a multi-node job, the measurements from all nodes used by the job are further aggregated to obtain the total job-level FLOPs and memory traffic. The job-level operational intensity is then calculated and compared with the Roofline threshold to assign the compute-bound or memory-bound label. Therefore, the labels are generated at the job level rather than at the individual-node or system level.

For each submitted job, the UHBF over the past preceding N -day window is computed based on the username and job submission time. This four-dimensional vector is then concatenated with the semantic representation of the job to construct a comprehensive feature vector, which is used as input to the XGBoost model. During model training, these generated labels are used as ground truth for model training and validation, thereby yielding the trained HPC job classification model.

3.5. Exponentially Weighted Moving Average (EWMA) Algorithm

Inspired by the methods proposed in [53,54], this paper introduces an exponentially weighted moving average (EWMA) method for HPC workloads. During the model inference stage, only the true labels of users' jobs completed before the submission time of the current job are used to update the short-term class distribution of user-submitted jobs through the EWMA algorithm, thereby obtaining a short-term probability estimate for each job type.

The proposed EWMA algorithm is presented in Algorithm 2. For each user or user–task–name combination, a corresponding template key is defined, and the associated drift state is initialized. This state includes the current probability estimate p_{t-1} and the number of observed completed jobs n_{obs} . When the drift state of a given key, such as a specific user or a user–task–name combination, is required, the corresponding state information is retrieved from the state table using this key. If the key does not yet exist in the state table, a new state record is automatically created, with its probability initialized to $p_{init} = 0.5$ and its observation count set to 0. The drift states for different keys are maintained independently to track behavioral changes across users or user–task–name combinations. The EWMA update coefficient α is introduced to control the balance between historical probability estimates and new observations. A larger α makes the estimate more sensitive to recent class changes, while a smaller α produces smoother and more stable probability estimates.

When a new job is submitted to the system, the current probability estimate p_{t-1} corresponding to its key is retrieved and used as the short-term probability estimate of the job type distribution. After the job is completed and its true label becomes available, the EWMA algorithm is applied to update the probability estimate as follows:

$$p_t = (1 - \alpha)p_{t-1} + \alpha y \quad (6)$$

where $y \in \{0, 1\}$ denotes the true class label of the completed job, with 0 representing memory-bound and 1 representing compute-bound, and p_t represents the updated short-term probability estimate. This update mechanism allows the latest observation to influence the probability estimate while preserving historical information, thereby smoothly tracking changes in the job class distribution. After the update, the observation count is incremented as follows: $n_{obs} = n_{obs} + 1$.

Algorithm 2: Exponentially Weighted Moving Average (EWMA)

Input: Template identifier *key* (user or user–task–name combination), true label y of the completed job, initial probability p_{init} , state table S , smoothing coefficient α

Output: Updated short-term probability estimate p_t and observation count n_{obs}

1. If the state corresponding to *key* does not exist in S , initialize:

$$S[key].p \leftarrow p_{init}$$

$$S[key].n_{obs} \leftarrow 0$$
 2. Retrieve the current state of *key* :

$$p_{t-1} \leftarrow S[key].p$$

$$n_{obs} \leftarrow S[key].n_{obs}$$
 3. After the job is completed and its true label y becomes available, update the probability estimate:

$$p_t \leftarrow (1 - \alpha)p_{t-1} + \alpha y$$
 4. Update the observation count:

$$n_{obs} \leftarrow n_{obs} + 1$$
 5. Write the updated state back to the state table:

$$S[key].p \leftarrow p_t$$

$$S[key].n_{obs} \leftarrow n_{obs}$$
 6. Return p_t, n_{obs}
-

3.6. Probability-Weighted Fusion Algorithm

Based on the EWMA probability and the number of observations corresponding to the current user or user–task–name combination, the drift weight is dynamically computed using a warm-up threshold. To mitigate the excessive influence of the drift signal on

the final prediction when only a few observations are available during the initial stage, a warm-up mechanism is introduced, whereby the drift weight gradually increases with the number of observations n_{obs} . The weight is defined as:

$$w = \begin{cases} w_{max} \cdot \frac{n_{obs}}{warmup}, & n_{obs} < warmup \\ w_{max}, & n_{obs} \geq warmup \end{cases} \quad (7)$$

where *warmup* denotes the warm-up threshold, and w_{max} represents the maximum weight of the drift signal. When $n_{obs} < warmup$, the weight w increases linearly with the number of observations; when $n_{obs} \geq warmup$, the weight is fixed at w_{max} . This mechanism reduces the impact of the drift signal during the initial stage and gradually enhances its contribution to the probability fusion as more observations become available.

The proposed probability-weighted fusion algorithm is presented in Algorithm 3. When a user submits a job, the job description is first encoded using a PLM. Meanwhile, the UHBF vector over the past N days is extracted based on the submission time. The job semantic features and the UHBF vector are then concatenated to construct a comprehensive feature vector, which is fed into the HPC job classification model to obtain the predicted probabilities for different classes. The short-term EWMA estimate obtained for the corresponding key is then combined with the model output probability through weighted fusion to produce the final prediction probability:

$$p_{final} = (1 - w)p_{model} + wp_t \quad (8)$$

where p_{model} denotes the probability output by the classification model. The final prediction probability p_{final} is used for classification: if $p_{final} \geq 0.5$, the job is classified as class 1; otherwise, it is classified as class 0.

To prevent future information leakage, during the online evaluation process described in Algorithms 2 and 3, all test jobs are globally sorted by submission time. Before predicting the current job, only the ground-truth labels of historical jobs whose completion times are earlier than the submission time of the current job are used to update the corresponding EWMA state. The predicted probability output by the classification model is then fused with the current EWMA probability estimate to obtain the final prediction for the current job. After a job is completed and its ground-truth class label (y) becomes available, its completion time and ground-truth label are recorded under the key associated with the job and stored in an update queue for cache management. When a subsequent job associated with the same key is submitted, records satisfying the temporal condition are retrieved from the update queue, and their ground-truth class labels are used to update the EWMA state. Neither the ground-truth label of the current job itself nor those of other unfinished jobs are used for the current prediction. This ensures that no future label information is used during the online evaluation process.

Algorithm 3: Probability-Weighted Fusion Algorithm

Input: Template identifier key_i , classification model M , state table S , warm-up threshold *warmup*, job semantic feature vector h_i , UHBF vector f_i , maximum weight w_{max}

Output: Final prediction probability p_{final} , predicted label $\hat{y}_i$

1. Construct the feature vector:

$$z_i \leftarrow [h_i; f_i]$$

2. Obtain the model prediction:

$$p_{model} \leftarrow M(z_i)$$

Algorithm 3: *Cont.*

3. Retrieve the EWMA state from the state table:

$$(p_t, n_{obs}) \leftarrow S[key_i]$$

4. Compute drift weight:

$$\text{if } n_{obs} < warmup, \text{ then } w \leftarrow w_{max} \cdot \frac{n_{obs}}{warmup} \\ \text{else } w \leftarrow w_{max}$$

5. Probability-weighted fusion:

$$p_{final} \leftarrow (1 - w)p_{model} + wp_t$$

6. Classification decision:

$$\text{if } p_{final} \geq 0.5, \text{ then } \hat{y}_i \leftarrow 1 \\ \text{else } \hat{y}_i \leftarrow 0$$

7. Return $p_{final}, \hat{y}_i$

In the proposed DA-HJSC framework, during the model inference stage, the short-term class probabilities estimated by EWMA are fused with the probabilities predicted by the HPC job classification model, enabling online adaptive classification of job types. The main technical contribution of this study lies not in the individual use of a PLM, XGBoost, or EWMA, but rather in modeling users' long-term behavioral patterns through UHBF and updating the EWMA-based short-term class probabilities using the ground-truth labels of recently completed jobs. This mechanism dynamically updates the class probabilities as jobs are submitted and completed, without requiring a fixed short-term sliding time window. Through the coordinated use of UHBF and EWMA, the framework captures both long-term patterns and short-term fluctuations in users' job submission behavior, thereby mitigating the degradation in classification performance caused by concept drift.

4. Experiments

4.1. Dataset and Experimental Settings

To evaluate the proposed method, this study employs the F-DATA job dataset [32], which was collected from the Fugaku supercomputer. Fugaku (Fujitsu Limited, Kawasaki, Japan) is a production high-performance computing system deployed by the RIKEN Center for Computational Science in 2020, consisting of approximately 160 k compute nodes. Each node is equipped with 48 Arm compute cores, 4 assistant cores, and 32 GB of HBM2 memory with a bandwidth of 1024 GB/s, delivering a peak performance of approximately 3.3 TFlops/s per node and an overall system peak performance of about 537 PFlops/s.

The F-DATA dataset is designed for HPC workload modeling and includes both job performance metrics and user-submitted job information. Sensitive information in the dataset has been irreversibly anonymized, making it suitable for job-level prediction and modeling tasks. In this study, the selected data span from 1 May 2023 to 30 April 2024. After data preprocessing, the dataset contains approximately 5.99 million job records. The performance metrics include job submission time, start time, end time, the number of allocated nodes, memory bandwidth, and floating-point operations. The user-submitted job information includes the username, task name, and the requested hardware resource information at submission time, such as the number of CPU cores, the CPU frequency, the number of nodes, and the runtime environment.

All experiments were conducted using Python 3.11.7, with Visual Studio Code 1.111.0 as the development environment. The experiments were performed on a machine equipped with an NVIDIA GeForce RTX 4070 GPU, an Intel Core i9-14900HX processor with 24 cores, and 32 GB of RAM.

4.2. Pre-Trained Language Models

To evaluate the effectiveness of the proposed method, three PLMs are employed to encode job descriptions, namely, all-MiniLM-L6-v2 [55], bge-base-en-v1.5 [56], and e5-base-v2 [57]. These models are Transformer-based encoder-only models capable of capturing contextual semantic information and providing high-quality embeddings for downstream tasks. Among them, all-MiniLM-L6-v2 is a lightweight model with a 6-layer architecture that generates 384-dimensional embeddings and offers high encoding efficiency. In contrast, both bge-base-en-v1.5 and e5-base-v2 adopt 12-layer architectures and produce 768-dimensional embeddings. Specifically, bge-base-en-v1.5 is designed for high-quality text representation in retrieval tasks, while e5-base-v2 serves as a general-purpose embedding model. By employing different PLMs for job description encoding, we aim to evaluate the robustness and generalizability of the proposed DA-HJSC framework under different semantic representation settings.

4.3. Comparative Methods

Random Forest (RF) and Adaptive Random Forest (ARF) are selected as comparative methods to evaluate the applicability of the DA-HJSC framework to different classifiers and to compare the performance of different drift-adaptation mechanisms, respectively. For the RF method, the original XGBoost classifier in the DA-HJSC framework is replaced with RF, while all other components and experimental settings remain unchanged. By comparing the experimental results of the XGBoost- and RF-based DA-HJSC methods, we examine the applicability of the framework to different classification models and analyze the influence of classifier selection on HPC job classification performance. ARF is employed as a comparative method with online incremental learning and concept drift adaptation capabilities. It is compared with the XGBoost baseline and the XGBoost- and RF-based DA-HJSC methods to evaluate the classification accuracy, performance stability, and online adaptability of different methods in dynamic HPC job classification scenarios. ARF is first warm-started using completed jobs from the 30 days preceding the beginning of the test period. It then performs online prediction in chronological order by job submission time and is incrementally updated using only jobs that have been completed and whose ground-truth label becomes available before the submission time of the job being predicted, thereby preventing the use of future information. ARF uses only job semantic embeddings as input and does not incorporate the UHBF or EWMA components proposed in this study. Although ARF and the XGBoost baseline use the same semantic embeddings, the direct comparison between ARF and the complete DA-HJSC framework is not fully feature-controlled because DA-HJSC additionally incorporates UHBF. Therefore, this comparison primarily reflects the overall performance of the complete methods.

4.4. Evaluation Metrics

This study adopts a rolling-window training-testing strategy for evaluation. Specifically, a continuous 30-day period is used as the training window, within which the data are further split into training and validation subsets with a ratio of 9:1. The trained model is then evaluated on the data from the subsequent day, i.e., the 31st day. This process slides the window forward day by day in chronological order, enabling continuous evaluation of the model's temporal generalization capability. During rolling model training, only historical jobs completed before the beginning of the corresponding test day are included in the training data, ensuring that their ground-truth labels are already available at training time. During daily testing, all test jobs are processed in chronological order according to their submission times. At the beginning of each test day, the EWMA state is reinitialized, and no state is carried over from the previous test day. Before predicting the current

job, the EWMA state is updated only using test jobs completed on that test day whose ground-truth labels have become available before the submission time of the current job. The ground-truth label of the current job is not provided to the algorithm during prediction and can be used for subsequent EWMA updates only after the job has been completed. These temporal constraints ensure that both model training and online EWMA updates strictly follow the principle of causality and do not use the current label or any unavailable future information. The parameter settings used in the experiments are as follows: $N = 90$, $\alpha = 0.05$, $warmup = 50$, and $w_{max} = 0.8$.

The primary evaluation metric is the macro-averaged $F1$ score, which is used to assess the classification performance for compute-bound and memory-bound jobs. The $F1$ score for a single class is defined as the harmonic mean of precision and recall:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (9)$$

The $F1_{macro}$ score is defined as the arithmetic mean of the $F1$ scores of all classes:

$$F1_{macro} = \frac{F1^{(0)} + F1^{(1)}}{2} \quad (10)$$

After completing the entire rolling training–testing process, the model is evaluated from both temporal and overall perspectives. First, the daily $F1_{macro}$ score is computed for each test day to analyze the temporal variation and stability of model performance over time. Second, all predictions across the entire test period are aggregated to compute the $F1_{macro}$ (overall), which is used to evaluate the aggregate classification performance of the model.

Additional evaluation metrics include per-class precision, recall, and $F1$ score. The mean, standard deviation, variance, maximum, and minimum of the daily $F1_{macro}$ scores are also reported to comprehensively evaluate the classification performance and temporal stability of the different methods. Furthermore, based on the daily $F1_{macro}$ scores obtained on the same test dates, p -values are calculated using a paired two-sided Wilcoxon signed-rank test. The XGBoost-based DA-HJSC method is used as the reference and compared pairwise with the other methods. A performance difference between two methods is considered statistically significant when the p -value is less than 0.05.

4.5. Experimental Results

In this study, the DA-HJSC framework used XGBoost as the default base classifier, while RF was also used in place of XGBoost to evaluate the framework's applicability to different classifiers. Three PLMs were selected for semantic encoding, and experiments with different configurations were conducted to comprehensively assess the classification performance and temporal stability of DA-HJSC on the dataset. A rolling evaluation strategy was adopted from 1 June 2023 to 30 April 2024, spanning 335 calendar days, of which seven contained no valid job records. As shown in Table 1, four experimental strategies are evaluated for each combination of classifier and PLM, namely, baseline, +UHBF, +EWMA, and +UHBF+EWMA. Baseline uses only job semantic features; +UHBF additionally incorporates user historical behavioral features; +EWMA applies EWMA-based probability-weighted fusion to the model-predicted probabilities; and +UHBF+EWMA combines both mechanisms.

Based on the baseline results, XGBoost achieves slightly better overall classification performance than RF, and both classifiers obtain relatively strong performance when bge-base-en-v1.5 is used for semantic encoding. Specifically, XGBoost achieves an $F1_{macro}$ (overall) of 0.9010, with 17 low-performance days ($F1_{macro} < 0.6$) and 46 medium-performance days

($0.6 \leq F1_{macro} < 0.8$). RF achieves an $F1_{macro}$ (overall) of 0.8994, with 18 low-performance days and 50 medium-performance days. These results indicate that the baseline performance obtained using the three semantic encoding models is generally similar across the two classifiers.

Table 1. Overall Classification Performance of Different Methods across Classifiers and Pre-Trained Language Models.

Classifier	PLM	Method	$F1_{macro}$ (Overall)	Accuracy	Class 0 F1 Score	Class 1 F1 Score	$F1_{macro} < 0.6$ (Days)	$0.6 \leq F1_{macro}$ < 0.8 (Days)
XGBoost	all-MiniLM-L6-v2	Baseline	0.8997	0.9227	0.9478	0.8517	15	54
		+UHBF	0.9009	0.9222	0.9468	0.8550	14	35
		+EWMA	0.9222	0.9403	0.9598	0.8846	7	36
		+UHBF+EWMA	0.9272	0.9436	0.9618	0.8926	7	19
	bge-base-en-v1.5	Baseline	0.9010	0.9235	0.9482	0.8538	17	46
		+UHBF	0.9023	0.9237	0.9480	0.8567	15	40
		+EWMA	0.9226	0.9404	0.9597	0.8855	10	32
		+UHBF+EWMA	0.9250	0.9419	0.9605	0.8895	8	24
	e5-base-v2	Baseline	0.9009	0.9237	0.9485	0.8533	16	51
		+UHBF	0.8999	0.9215	0.9464	0.8533	15	36
		+EWMA	0.9225	0.9406	0.9599	0.8852	10	31
		+UHBF+EWMA	0.9263	0.9429	0.9612	0.8914	6	20
RF	all-MiniLM-L6-v2	Baseline	0.8980	0.9225	0.9480	0.8481	16	55
		+UHBF	0.9103	0.9313	0.9537	0.8669	12	44
		+EWMA	0.9193	0.9386	0.9588	0.8798	8	36
		+UHBF+EWMA	0.9257	0.9430	0.9616	0.8898	8	31
	bge-base-en-v1.5	Baseline	0.8994	0.9234	0.9485	0.8503	18	50
		+UHBF	0.9093	0.9306	0.9533	0.8653	13	45
		+EWMA	0.9209	0.9398	0.9595	0.8822	10	36
		+UHBF+EWMA	0.9248	0.9424	0.9612	0.8883	7	36
	e5-base-v2	Baseline	0.8977	0.9223	0.9479	0.8474	19	52
		+UHBF	0.9087	0.9303	0.9531	0.8643	13	45
		+EWMA	0.9193	0.9388	0.9589	0.8797	11	36
		+UHBF+EWMA	0.9245	0.9423	0.9612	0.8878	8	31

After incorporating UHBF, the $F1_{macro}$ (overall) scores of the XGBoost models using all-MiniLM-L6-v2 and bge-base-en-v1.5 both increase. In particular, the score of the bge-base-en-v1.5-based method increases from 0.9010 to 0.9023. However, the score of the e5-base-v2-based method decreases slightly from 0.9009 to 0.8999. Although the changes in $F1_{macro}$ (overall) are relatively small, the number of low- and medium-performance days generally decreases for all three encoding models. By comparison, RF achieves relatively consistent performance improvements across all three semantic encoding models. The most notable improvement is observed for the all-MiniLM-L6-v2-based method, whose $F1_{macro}$ (overall) increases from 0.8980 to 0.9103, while the number of low-performance days decreases from 16 to 12 and the number of medium-performance days decreases from 55 to 44. These results indicate that UHBF can provide complementary information about users' long-term behavioral patterns, although its actual effectiveness is influenced by the choice of base classifier and semantic encoding model.

After introducing EWMA-based probability fusion alone during the model inference stage, both XGBoost and RF achieve substantial performance improvements across the three semantic encoding models. For XGBoost, the methods using all-MiniLM-L6-v2, bge-base-en-v1.5, and e5-base-v2 achieve $F1_{macro}$ (overall) scores of 0.9222, 0.9226, and 0.9225, respectively, corresponding to improvements of 2.25, 2.16, and 2.16 percentage points over their respective baselines. The number of low-performance days decreases from 15, 17, and 16 to 7, 10, and 10, respectively, while the number of medium-performance days decreases from 54, 46, and 51 to 36, 32, and 31, respectively. For RF, the methods using the three semantic encoding models achieve $F1_{macro}$ (overall) scores of 0.9193, 0.9209, and 0.9193,

respectively, representing improvements of 2.13, 2.15, and 2.16 percentage points over their respective baselines. The number of low-performance days decreases to 8, 10, and 11, respectively, while the number of medium-performance days decreases to 36 for all three models. These results demonstrate that EWMA can dynamically adjust the prediction results according to recent changes in the class distribution of completed jobs, thereby effectively mitigating performance fluctuations caused by short-term concept drift.

When UHBF and EWMA are incorporated simultaneously, all six combinations of classifiers and PLMs achieve their respective highest $F1_{macro}$ (overall) scores. For XGBoost, the DA-HJSC method using all-MiniLM-L6-v2 performs best, achieving an $F1_{macro}$ (overall) score of 0.9272, which is 2.75 percentage points higher than the corresponding baseline. Its accuracy reaches 0.9436, representing an improvement of 2.09 percentage points, while its $F1$ score for class 1 reaches 0.8926, an improvement of 4.09 percentage points. Meanwhile, the number of low-performance days decreases from 15 to 7, and the number of medium-performance days decreases from 54 to 19. The DA-HJSC methods using bge-base-en-v1.5 and e5-base-v2 achieve $F1_{macro}$ (overall) scores of 0.9250 and 0.9263, respectively. Among them, the e5-base-v2-based method further reduces the number of low-performance days to 6. For RF, the methods using all-MiniLM-L6-v2, bge-base-en-v1.5, and e5-base-v2 achieve $F1_{macro}$ (overall) scores of 0.9257, 0.9248, and 0.9245, respectively, corresponding to improvements of 2.77, 2.54, and 2.68 percentage points over their respective baselines. In particular, the all-MiniLM-L6-v2-based method achieves an accuracy of 0.9430 and an $F1$ score of 0.8898 for class 1. The bge-base-en-v1.5-based method reduces the number of low-performance days from 18 to 7, while the e5-base-v2-based method reduces it from 19 to 8.

Overall, XGBoost performs slightly better than RF in terms of $F1_{macro}$ (overall), the $F1$ score for class 1, and other metrics. Nevertheless, the RF-based DA-HJSC methods also achieve consistent improvements across the three semantic encoding models, demonstrating the strong robustness of the framework. In this experiment, UHBF is primarily used to characterize users' long-term behavioral patterns, whereas EWMA is employed to adapt to short-term changes in class distributions. Their combination further improves overall classification performance, enhances the identification of compute-bound jobs, and reduces the occurrence of low- and medium-performance days. These results demonstrate the effectiveness and stability of DA-HJSC in addressing dynamic HPC workloads and concept drift.

Using XGBoost as the classifier, Figures 2–4 present the variations in daily $F1_{macro}$ for different classification methods based on semantic representations generated by different PLMs. The thin, light-colored lines represent the daily $F1_{macro}$ scores, whereas the thick, dark-colored lines represent the corresponding $F1_{macro}$ scores smoothed using a 7-day moving average. Over time, the daily classification performance of all methods exhibits varying degrees of fluctuation, but clear differences are observed in their overall performance levels and fluctuation magnitudes.

As shown in Figures 2–4, compared with the baseline, +UHBF, and +EWMA methods, DA-HJSC achieves better daily classification performance under all three PLMs, with $F1_{macro}$ (overall) scores of 0.9272, 0.9250, and 0.9263, respectively. During July and October 2023 and March 2024, DA-HJSC improves classification performance and mitigates the decline in the 7-day moving-average curves. Together with the statistical results in Table 1, these observations show that DA-HJSC reduces the number of low- and medium-performance days and maintains model performance at approximately 0.9 or above on most test days. During June, November, and December 2023 and January 2024, all methods generally maintain high performance, and DA-HJSC also exhibits similar performance across the three PLMs. Even when the baseline model already achieves a high daily $F1_{macro}$

score, DA-HJSC can still reduce daily performance fluctuations to some extent and provide more stable performance. This finding indicates that UHBF and EWMA are complementary. UHBF characterizes relatively long-term user behavioral patterns, whereas EWMA tracks recent changes in the class distribution of completed jobs. Their joint use helps achieve a better balance between overall performance and temporal stability. In April 2024, the daily classification performance of all methods exhibits relatively pronounced fluctuations, with $F1_{macro}$ falling below 0.8 on some days. The declines in both the daily $F1_{macro}$ trend and the 7-day moving-average curve of DA-HJSC are generally smaller than those of the baseline and +UHBF methods, indicating that DA-HJSC can alleviate low- and medium-performance fluctuations to a certain extent. This phenomenon suggests that HPC jobs are highly nonstationary on a daily time scale, which is likely related to temporal changes in user job behavior, workload composition, and resource usage patterns. When the data distribution deviates from that observed during model training, classification performance may decline abruptly. During August and September 2023 and February 2024, the results of DA-HJSC and the other methods are relatively similar on some days, and the improvements achieved by DA-HJSC during certain low- and medium-performance periods remain limited. All methods with the three PLMs exhibit noticeable performance degradation, indicating that DA-HJSC still has certain limitations when the daily class distribution is highly imbalanced or the drift pattern is relatively complex. Overall, DA-HJSC demonstrates favorable classification performance and temporal stability across different pre-trained language models.

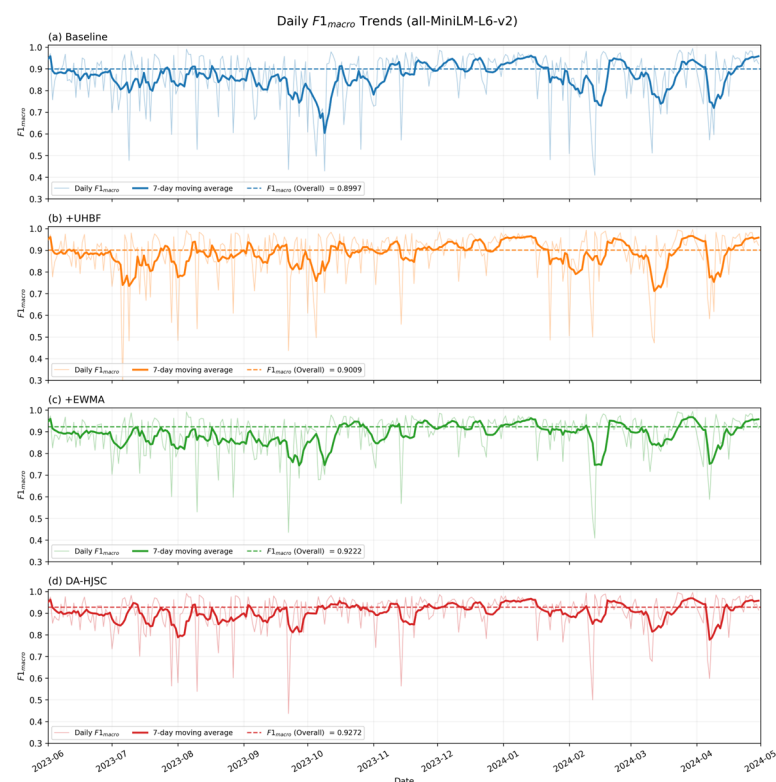


Figure 2. HPC Job Classification Performance of Different Methods Using all-MiniLM-L6-v2 Embeddings and XGBoost. The subfigures show (a) the baseline, (b) the baseline with UHBF, (c) the baseline with EWMA, and (d) DA-HJSC.

The daily sample size and class distribution of the test set are presented in Table 2. The test set contains a total of 5,467,969 jobs over the entire test period, including 4,014,373 memory-bound jobs, accounting for approximately 73.42%, and 1,453,596 compute-bound jobs, accounting for approximately 26.58%. The number of

memory-bound jobs is approximately 2.76 times that of compute-bound jobs, indicating a certain degree of class imbalance in the test set. The average daily sample size is 16,671 jobs, with a standard deviation of 13,039, indicating substantial variation in the number of samples across different test days. The daily number of memory-bound jobs ranges from 11 to 64,660, whereas the daily number of compute-bound jobs ranges from 0 to 69,267. These results indicate that the class distribution is more severely imbalanced on certain days. Based on the sample composition and class distribution of the test set, we further investigated the influence of different feature groups on the model's classification decisions. Taking the bge-base-en-v1.5 encoder as an example, we conducted a TreeSHAP analysis of the XGBoost-based method using both semantic features and UHBF. We aggregated the absolute TreeSHAP values for 5,467,969 samples in the test set by feature group and normalized the aggregated values across the semantic feature and UHBF groups. The results show that semantic features accounted for 68.97% of the total feature attribution of the XGBoost classifier, whereas UHBF accounted for 31.03%. These findings indicate that the model did not rely excessively on either feature group. Semantic representations remained the primary information source for job classification, while the four-dimensional UHBF vector made a substantial complementary contribution, thereby supporting the rationale for fusing the two feature groups.

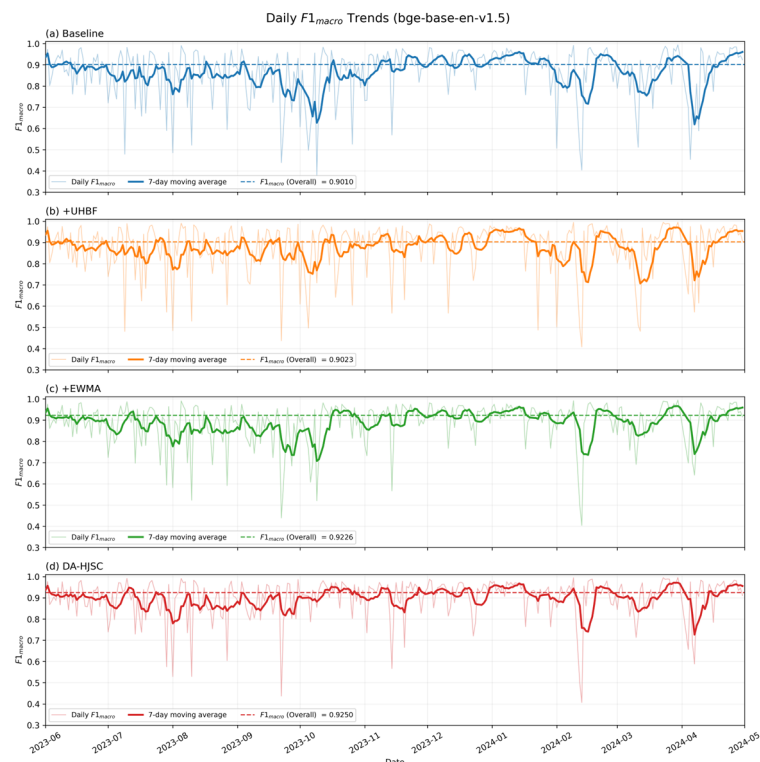


Figure 3. HPC Job Classification Performance of Different Methods Using bge-base-en-v1.5 Embeddings and XGBoost. The subfigures show (a) the baseline, (b) the baseline with UHBF, (c) the baseline with EWMA, and (d) DA-HJSC.

In addition, the proposed DA-HJSC method is compared with the drift-adaptive ARF method, and the corresponding results are presented in Table 3. Using the bge-base-en-v1.5 as an example, ARF achieves an $F1_{macro}$ (overall) score of 0.8263. Its recall for class 0 reaches 0.9792, whereas the recall for class 1 is only 0.6071. The confusion matrix further shows that ARF incorrectly classifies 571,128 true class 1 jobs as class 0, accounting for 39.29% of all class 1 jobs. This indicates that ARF can effectively identify memory-bound jobs but has relatively limited capability in recognizing compute-bound jobs. Under the experimental settings of

this study, ARF, despite its online incremental learning and drift-adaptation capabilities, still exhibits certain limitations in adapting to complex and dynamically changing HPC job patterns. By contrast, both the baseline and DA-HJSC achieve $F1_{macro}$ (overall) scores above 0.90, with class 1 recall values exceeding 0.84. Among them, DA-HJSC using the XGBoost classifier achieves the highest class 1 recall and $F1$ score, reaching 0.8803 and 0.8895, respectively. Compared with the baseline, the number of true class 1 jobs incorrectly classified as class 0 decreases from 232,218 to 174,014, demonstrating that the proposed method can effectively improve the recognition of compute-bound jobs. DA-HJSC using the RF classifier also achieves favorable results, with an $F1_{macro}$ (overall) score of 0.9248, indicating that the proposed framework is applicable to different base classifiers.

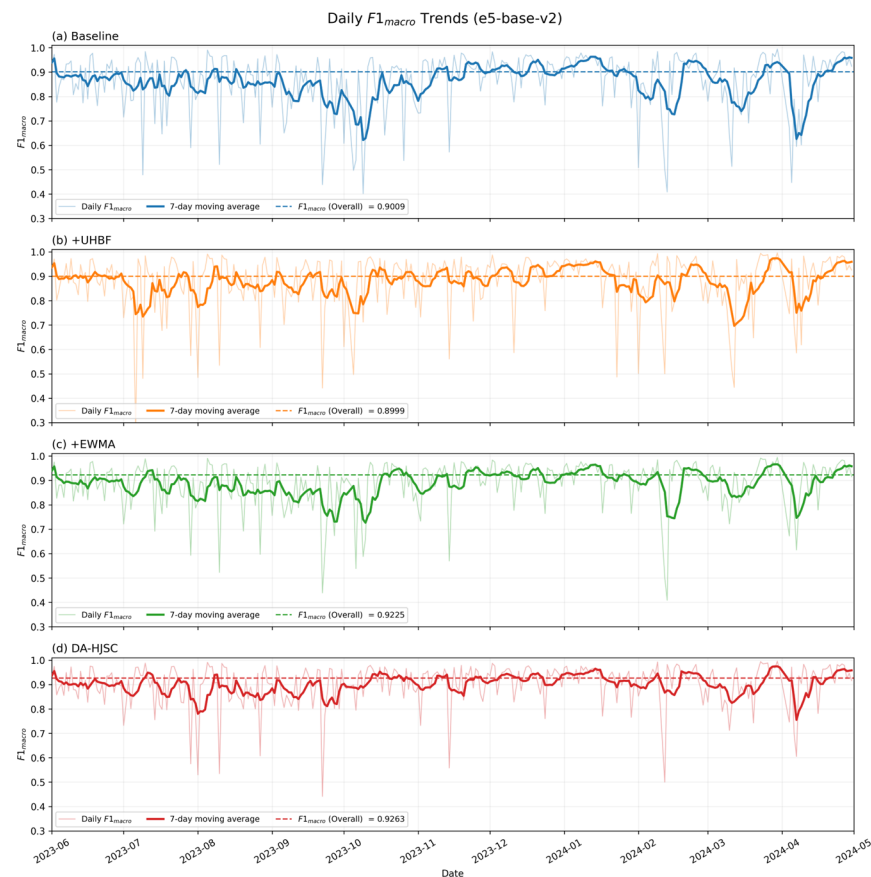


Figure 4. HPC Job Classification Performance of Different Methods Using e5-base-v2 Embeddings and XGBoost. The subfigures show (a) the baseline, (b) the baseline with UHBF, (c) the baseline with EWMA, and (d) DA-HJSC.

Table 2. Daily Sample Statistics and Class Distribution in the Test Set.

Class	Daily Sample Avg.	Daily Sample Std.	Daily Sample Max	Daily Sample Min	Total Samples
Total	16,671	13,039	88,120	11	5,467,969
Class 0	12,239	8903	64,660	11	4,014,373
Class 1	4432	6651	69,267	0	1,453,596

Table 4 further presents the daily performance stability and statistical significance results of the different methods. Using the bge-base-en-v1.5 as an example, the XGBoost-based DA-HJSC method achieves the highest mean, maximum, and minimum daily $F1_{macro}$ scores, with values of 0.9006, 0.9955, and 0.4067, respectively. It also yields the lowest standard deviation and variance, with values of 0.0905 and 0.0082, respectively. These results

indicate that the method not only achieves higher average daily classification performance but also exhibits smaller performance fluctuations. Using the XGBoost-based DA-HJSC method as the reference, the paired two-sided Wilcoxon signed-rank test shows that the p -values for its comparisons with ARF and baseline are both less than 0.0001, while the p -value for its comparison with the RF-based DA-HJSC method is 0.0002. All p -values are below the significance level of 0.05, indicating that the differences in daily $F1_{macro}$ between the reference method and each comparison method are statistically significant. Compared with the RF-based DA-HJSC method, the XGBoost-based DA-HJSC method increases the mean daily $F1_{macro}$ from 0.8940 to 0.9006, representing an improvement of 0.66 percentage points. These results demonstrate that, compared with ARF and baseline, DA-HJSC achieves higher overall classification performance and average daily performance while reducing performance fluctuations and improving classification accuracy. In particular, when XGBoost is used as the base classifier, the method shows better recognition performance for compute-bound jobs, indicating that DA-HJSC can effectively adapt to dynamic changes in HPC job patterns.

Table 3. Per-Class Performance Metrics of Different Methods Using bge-base-en-v1.5.

Method	Classifier	$F1_{macro}$ (Overall)	Class	Precision	Recall	F1 Score	Predicted Class 0	Predicted Class 1
ARF	ARF	0.8263	0	0.8731	0.9792	0.9231	3,930,855 (97.92%)	83,518 (2.08%)
			1	0.9135	0.6071	0.7294	571,128 (39.29%)	882,468 (60.71%)
Baseline	XGBoost	0.9010	0	0.9428	0.9536	0.9482	3,828,275 (95.36%)	186,098 (4.64%)
			1	0.8678	0.8402	0.8538	232,218 (15.98%)	1,221,378 (84.02%)
DA-HJSC	XGBoost	0.9250	0	0.9570	0.9641	0.9605	3,870,450 (96.41%)	143,923 (3.59%)
			1	0.8989	0.8803	0.8895	174,014 (11.97%)	1,279,582 (88.03%)
	RF	0.9248	0	0.9509	0.9717	0.9612	3,900,767 (97.17%)	113,606 (2.83%)
			1	0.9168	0.8616	0.8883	201,210 (13.84%)	1,252,386 (86.16%)

Table 4. Daily $F1_{macro}$ Stability and Statistical Significance Results Using bge-base-en-v1.5.

Method	Classifier	Daily $F1_{macro}$ Mean	Daily $F1_{macro}$ Std.	Daily $F1_{macro}$ Var.	Daily $F1_{macro}$ Max	Daily $F1_{macro}$ Min	p -Value
ARF	ARF	0.7563	0.1318	0.0174	0.9916	0.3781	<0.0001
Baseline	XGBoost	0.8683	0.1171	0.0137	0.9945	0.3785	<0.0001
DA-HJSC	XGBoost	0.9006	0.0905	0.0082	0.9955	0.4067	—
	RF	0.8940	0.0928	0.0086	0.9951	0.4036	0.0002

Using bge-base-en-v1.5 as an example, the sensitivity of DA-HJSC to the UHBF historical window size was first analyzed. After the window size was fixed, the EWMA-related parameters were selected using validation data, and their sensitivity was subsequently evaluated. The results are presented in Table 5. Five candidate UHBF historical window sizes were considered: 14, 30, 60, 90, and 120 days. Because the dataset begins on 1 May 2023, only approximately 30 days of historical data were available before the final test period. Therefore, the initial validation set did not provide sufficient historical coverage to adequately compare longer windows, such as 60, 90, and 120 days. To ensure that all candidate windows were compared over the same entire rolling test period, an exploratory sensitivity analysis for all candidate windows was conducted using data from that test period. Because this analysis did not use independent validation data, selecting the window based on these results may introduce some parameter-selection bias. When the window size is set to 90 days, the model achieves the highest $F1_{macro}$ (overall) and mean daily $F1_{macro}$ scores, reaching 0.9250 and 0.9006, respectively, while the number of medium-performance

days decreases to 24. The number of low-performance days remains relatively stable across different window sizes, ranging from 7 to 8, whereas the number of medium-performance days varies more noticeably. Although the 120-day window yields the lowest standard deviation of daily $F1_{macro}$, at 0.0879, its $F1_{macro}$ (overall) score and number of medium-performance days are both inferior to those obtained with the 90-day window. Overall, the 90-day window provides a better balance between users' long-term behavioral patterns and recent behavioral changes. Notably, the differences in $F1_{macro}$ (overall) among the 60-, 90-, and 120-day windows do not exceed 0.0016, indicating that DA-HJSC is not highly sensitive to the specific choice among these longer historical windows.

Table 5. One-at-a-time sensitivity analysis of DA-HJSC parameters on the test set using bge-base-en-v1.5.

Parameter	Value	$F1_{macro}$ (Overall)	Daily $F1_{macro}$ Mean	Daily $F1_{macro}$ Std.	$F1_{macro} < 0.6$ (Days)	$0.6 \leq F1_{macro}$ < 0.8 (Days)
UHBF Window Size	14 d	0.9230	0.8967	0.0900	7	31
	30 d	0.9220	0.8960	0.0928	8	27
	60 d	0.9234	0.8989	0.0894	7	25
	90 d	0.9250	0.9006	0.0905	8	24
	120 d	0.9238	0.8990	0.0879	8	28
EWMA Coefficient α	0.005	0.9218	0.8960	0.0923	8	32
	0.01	0.9232	0.8984	0.0917	8	28
	0.05	0.9250	0.9006	0.0905	8	24
	0.1	0.9247	0.9008	0.0900	9	23
	0.2	0.9243	0.9009	0.0899	9	24
<i>warmup</i> Threshold	20	0.9231	0.8961	0.0925	9	29
	30	0.9240	0.8982	0.0912	8	29
	50	0.9250	0.9006	0.0905	8	24
	100	0.9229	0.8977	0.0923	8	30
	200	0.9220	0.8963	0.0939	8	31
Maximum Drift Weight w_{max}	0.2	0.9071	0.8819	0.1130	14	40
	0.5	0.9158	0.8902	0.1052	12	33
	0.8	0.9250	0.9006	0.0905	8	24
	1.0	0.9238	0.8983	0.0907	8	27
Roofline Threshold Coefficient	0.9	0.9234	0.9001	0.0881	6	28
	1.0	0.9250	0.9006	0.0905	8	24
	1.1	0.9241	0.9003	0.0911	8	32

After fixing the maximum UHBF historical window size at 90 days, the EWMA update coefficient α , the *warmup* threshold, and the maximum drift weight w_{max} were selected using a held-out validation subset comprising 10% of the data from the 30-day historical window preceding the first test day (1 June 2023). The validation jobs were replayed in chronological order of submission, and only jobs that had been completed and whose ground-truth labels had become available before the submission of the current job were used to update the EWMA state. Parameter selection was not based solely on the highest validation $F1_{macro}$. Instead, configurations within 0.005 of the highest value were considered to have comparable performance. Based on both validation performance and online drift-adaptation capability, α , the *warmup* threshold, and w_{max} were set to 0.05, 50, and 0.8, respectively. This configuration achieved an $F1_{macro}$ of 0.9731 on the validation set. The window size and these EWMA parameters were then kept fixed throughout the final test period.

Furthermore, because the final test set contains more job samples and covers a longer rolling test period, the effects of parameter variations on overall model performance and

temporal stability can be more fully examined with a larger sample size and over a longer time span. Therefore, after the parameters had been selected using the validation set and the parameter configuration had been fixed, a one-at-a-time sensitivity analysis was subsequently conducted on the final test set. It should be noted that the one-at-a-time sensitivity analysis on the final test set was conducted solely to evaluate the robustness of the EWMA parameters determined during the validation stage and was not used to reselect or adjust α , the *warmup* threshold, or w_{max} . As shown in Table 5, when α , *warmup*, and w_{max} are set to 0.05, 50, and 0.8, respectively, the model achieves the highest $F1_{macro}$ (overall) score of 0.9250. Specifically, $\alpha = 0.05$ provides a favorable overall balance between the $F1_{macro}$ (overall) and the number of low- and medium-performance days. When *warmup* is set to 50, the model outperforms the other settings in terms of the $F1_{macro}$ (overall), the mean daily $F1_{macro}$, and the number of medium-performance days, indicating that an excessively low or high *warmup* threshold may affect EWMA state estimation. By comparison, the maximum drift weight w_{max} has the most pronounced effect on model performance. As w_{max} increases from 0.2 to 0.8, $F1_{macro}$ (overall) increases from 0.9071 to 0.9250, the standard deviation of daily $F1_{macro}$ decreases from 0.1130 to 0.0905, the number of low-performance days decreases from 14 to 8, and the number of medium-performance days decreases from 40 to 24. When w_{max} is further increased to 1.0, the overall performance decreases slightly. These results indicate that a small weight is insufficient for EWMA to effectively adjust predictions in response to short-term changes in the class distribution, whereas an excessively large weight may weaken the contribution of the offline classification model.

Furthermore, given the complexity of the Fugaku cluster and the possibility that the Roofline threshold may affect the class boundary between compute-bound and memory-bound jobs, the threshold coefficient is set to 0.9, 1.0, and 1.1 across the three settings in the sensitivity analysis. Under all three settings, the model maintains an $F1_{macro}$ (overall) score above 0.9234 and a mean daily $F1_{macro}$ score above 0.9, indicating that the overall model performance varies only slightly within the tested threshold range. Although different threshold settings have some influence on the number of low- and medium-performance days, they do not cause a noticeable decline in overall performance. These results demonstrate that DA-HJSC is robust to moderate variations in the Roofline threshold.

Table 6 presents the computational overhead of the individual components of DA-HJSC for the configuration based on bge-base-en-v1.5. The average encoding time of the PLM is 0.5961 ms/job. UHBF is constructed by retrieving and processing pre-generated daily statistical templates over the specified historical time window, with an average processing time of 0.0874 ms/job. DA-HJSC adopts a daily rolling training and testing mechanism. The end-to-end daily retraining time, including data loading, UHBF construction, model fitting, and model saving, is 55.9397 ± 45.7960 s/day. After training, the batch inference time of the XGBoost classifier is 0.000841 ms/job, while the complete EWMA online processing time is 43.042 μ s / job, of which a single EWMA state update requires only 0.722 μ s/update. After combining UHBF construction, feature concatenation, classifier inference, and EWMA processing, the average online processing time is 0.131907 ms/job. The size of the XGBoost model artifact generated each day is 0.3543 ± 0.0296 MB/model, which is substantially smaller than those of RF (16.2558 ± 8.3380 MB/model) and ARF (68.9764 ± 30.9138 MB/model). The average latency for model loading and the first prediction is 626.6233 ± 90.4769 ms. Compared with the baseline, DA-HJSC requires an additional 110.04 ± 219.81 MB of memory during training, whereas the memory difference during online processing is only -0.59 ± 34.60 MB, indicating that its additional online memory overhead is negligible. These results show that the main additional overhead of DA-HJSC is concentrated in the offline and daily retraining stages, whereas UHBF retrieval, classifier inference, and EWMA-based adaptive processing introduce only limited

online time and memory overhead. Therefore, DA-HJSC is suitable for dynamic HPC job classification scenarios.

Table 6. Computational Overhead of DA-HJSC.

Metric	Value
PLM Embedding Time	0.5961 ms/job
Average UHBF Construction Time	0.0874 ms/job
End-to-End Daily Retraining Time	55.9397 ± 45.7960 s/day
Batch Classifier Inference Time	0.000841 ms/job
EWMA Online Processing Time	43.042 µs/job
EWMA Update Time	0.722 µs/update
Model Loading and First-Prediction Latency	626.6233 ± 90.4769 ms
Average Online Processing Time	0.131907 ms/job
Model Artifact Size	0.3543 ± 0.0296 MB/model
Additional Training Memory of DA-HJSC	110.04 ± 219.81 MB
Additional Online Memory of DA-HJSC	−0.59 ± 34.60 MB

5. Discussion

The experimental results show that, without UHBF and the EWMA-based probability-weighted fusion mechanism, the baseline models trained using XGBoost or RF with embeddings generated by different PLMs exhibit only minor differences in $F1_{macro}$ (overall) and have a similar number of low- and medium-performance days. This suggests that, although the three PLMs differ in embedding dimensionality and semantic representation, the static classification models primarily rely on the historical data distribution within the training window. When the distribution of HPC jobs changes during inference, these models cannot promptly adjust their prediction strategies in response to recent variations, resulting in substantial performance fluctuations or even abrupt performance degradation on certain test days. Although ARF supports online incremental learning and concept drift adaptation, its overall classification performance is considerably lower than that of the other methods. As shown in Table 3, under the bge-base-en-v1.5 configuration, ARF achieves high recall for memory-bound jobs but relatively low recall for compute-bound jobs, indicating that many compute-bound jobs are incorrectly classified as memory-bound. This result also suggests that even jobs corresponding to the same task submitted by the same user may exhibit different workload types at different times as the HPC cluster environment evolves. Because such changes do not necessarily follow a regular pattern, continuous incremental updating alone remains insufficient to fully adapt to the complex, nonstationary variations in HPC workloads.

As shown in Table 1, after incorporating UHBF, the RF classifier achieves relatively consistent improvements in $F1_{macro}$ (overall) across all three PLMs, whereas the improvement achieved by XGBoost using bge-base-en-v1.5 is relatively limited. Nevertheless, both classifiers show reductions in the number of low- and medium-performance days during inference. This indicates that UHBF can supplement semantic features by summarizing users' long-term behavioral patterns. However, UHBF is essentially a static feature constructed from a historical window and therefore has limited ability to adapt to abrupt short-term drift during inference. After EWMA-based probability-weighted fusion is introduced during model inference, both the XGBoost- and RF-based models achieve substantial performance improvements across all three PLMs, particularly by reducing the number of low- and medium-performance days. EWMA continuously updates the short-term class probability estimates using the ground-truth labels of recently completed jobs and assigns greater weights to more recent observations. This mechanism dynamically adjusts the

classification tendency in response to recent changes in user behavior, thereby improving adaptability to short-term concept drift.

When UHBF and EWMA are incorporated simultaneously, all six combinations of classifiers and PLMs achieve relatively strong overall classification performance, as shown in Table 1 and Figures 2–4. Compared with the corresponding baseline models, DA-HJSC not only improves the $F1_{macro}$ (overall) but also enhances the $F1$ score and recall for class 1, as shown in Table 3, while substantially reducing the number of low- and medium-performance days. These results indicate that UHBF and EWMA are complementary across different temporal scales. UHBF models users' relatively long-term behavioral patterns, whereas EWMA captures short-term changes reflected by recently completed jobs. Their joint use enables DA-HJSC to exploit both stable long-term information and dynamic short-term information, thereby achieving a better balance between classification accuracy and temporal stability. As shown in Tables 3 and 4, the XGBoost-based DA-HJSC method generally performs slightly better than the RF-based DA-HJSC method and also outperforms ARF in terms of the $F1_{macro}$ (overall), the $F1$ score for class 1, average daily performance, and other metrics. Meanwhile, the RF-based DA-HJSC method also achieves consistent improvements across all three PLMs. This indicates that the effectiveness of DA-HJSC is not limited to a single classifier and that the framework is applicable to different base classifiers. The results in Table 6 further show that, compared with the RF-based DA-HJSC method and ARF, the XGBoost-based DA-HJSC method has a smaller model artifact, lower online processing overhead, and lower deployment resource requirements. Therefore, XGBoost is more suitable as the default base classifier for DA-HJSC, and the resulting framework demonstrates favorable stability, robustness, and practical deployment potential in dynamic HPC job classification.

Overall, through the coordinated use of an XGBoost model, UHBF, and EWMA-based probability-weighted fusion, DA-HJSC provides a mechanism for online drift-adaptation in dynamic HPC job classification. When user behavior or job distributions change, DA-HJSC continuously updates short-term probability estimates with EWMA based on the ground-truth labels of recently completed jobs and fuses these estimates with the output of the classification model, enabling the framework to respond promptly to such changes. This mechanism improves overall classification performance, reduces daily performance fluctuations, and enhances the identification of compute-bound jobs. The online adaptation strategy preserves the predictive capability of the offline classification model while strengthening the framework's adaptability to abrupt short-term changes in HPC workload distributions, thereby demonstrating its potential for long-term deployment in HPC clusters.

6. Conclusions

This paper proposes DA-HJSC, an online drift-adaptive framework for semantic representation and classification of HPC jobs. The framework combines job semantic features with UHBF to train an offline classification model, thereby enhancing the model's ability to capture users' long-term behavioral patterns. During model inference, an EWMA mechanism is introduced to dynamically update the class information of recently completed jobs. The resulting short-term probability estimates are then fused with the probabilities output by the classification model, enabling online adaptation to short-term changes in user behavior. Compared with traditional static classification methods, DA-HJSC jointly utilizes long-term behavioral information and short-term dynamic information to adjust prediction results across different temporal scales, thereby improving the framework's adaptability to changes in HPC job distributions. Experimental results demonstrate that DA-HJSC improves HPC job classification performance across different PLMs and base

classifiers, achieving the highest $F1_{macro}$ (overall) score of 0.9272. Compared with the baseline models and ARF, DA-HJSC not only improves overall classification performance and the identification of compute-bound jobs but also reduces the number of low- and medium-performance days and decreases fluctuations in daily classification performance. Overall, the proposed framework enables automated online adaptation of the job classification process without significantly increasing online computational overhead or requiring manual intervention, thereby demonstrating its potential for long-term deployment in HPC clusters. Future work will further investigate methods for improving classification accuracy and online adaptability under highly imbalanced workload distributions and more complex drift patterns.

Author Contributions: Data curation, methodology, experiment analysis, writing—original draft preparation, X.X.; Conceptualization, supervision, writing—review and editing, G.G.; investigation, validation, visualization, G.G. and J.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The F-DATA dataset used in this study is publicly available at Zenodo: <https://zenodo.org/records/11467483> (accessed on 4 November 2025).

Acknowledgments: We express our sincere appreciation to the creators of F-DATA for publicly releasing the data, which greatly supported this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Arianyan, E.; Gholipour, N.; Maleki, D.; Ghorbani, N.; Sepahvand, A.; Goudarzi, P. A Systematic Review and Classification of HPC-Related Emerging Computing Technologies. *Electronics* **2025**, *14*, 2476. [CrossRef]
2. Al Jawarneh, I.M.; Rosa, L.; Venanzi, R.; Foschini, L.; Bellavista, P. Efficient Parallel Processing of Big Data on Supercomputers for Industrial IoT Environments. *Electronics* **2025**, *14*, 2626. [CrossRef]
3. Rossi, F.; Saponara, S. Edge HPC Architectures for AI-Based Video Surveillance Applications. *Electronics* **2024**, *13*, 1757. [CrossRef]
4. Raj, R.K.; Romanowski, C.J.; Impagliazzo, J.; Aly, S.G.; Becker, B.A.; Chen, J.; Ghafoor, S.; Giacaman, N.; Gordon, S.I.; Izu, C.; et al. High Performance Computing Education: Current Challenges and Future Directions. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, ITiCSE-WGR '20, Trondheim, Norway, 17–18 June 2020*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 51–74.
5. Hou, Z.; Shen, H.; Zhou, X.; Gu, J.; Wang, Y.; Zhao, T. Prediction of Job Characteristics for Intelligent Resource Allocation in HPC Systems: A Survey and Future Directions. *Front. Comput. Sci.* **2022**, *16*, 165107. [CrossRef]
6. Patel, T.; Wagenhauser, A.; Eibel, C.; Honig, T.; Zeiser, T.; Tiwari, D. What does Power Consumption Behavior of HPC Jobs Reveal?: Demystifying, Quantifying, and Predicting Power Consumption Characteristics. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, 18–22 May 2020*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2020; pp. 799–809.
7. Tanash, M.; Yang, H.; Andresen, D.; Hsu, W. Ensemble Prediction of Job Resources to Improve System Performance for Slurm-Based HPC Systems. In *Proceedings of the Practice and Experience in Advanced Research Computing 2021: Evolution Across All Dimensions, PEARC '21, Boston, MA, USA, 18–22 July 2021*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 1–8.
8. Gama, J.; Medas, P.; Castillo, G.; Rodrigues, P. Learning with Drift Detection. In *Proceedings of the Advances in Artificial Intelligence—SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, 29 September–1 October 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 286–295.
9. Farrell, S.; Emani, M.; Balma, J.; Drescher, L.; Drozd, A.; Fink, A.; Fox, G.; Kanter, D.; Kurth, T.; Mattson, P.; et al. MLPerf™ HPC: A Holistic Benchmark Suite for Scientific Machine Learning on HPC Systems. In *Proceedings of the 2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC), St. Louis, MO, USA, 15 November 2021*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2021; pp. 33–45.
10. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76. [CrossRef]

11. Zhang, Y.; Ji, Z. TPSTA: A Tissue P System-Inspired Task Allocator for Heterogeneous Multi-Core Systems. *Electronics* **2026**, *15*, 1339. [\[CrossRef\]](#)
12. Ilic, A.; Pratas, F.; Sousa, L. Cache-Aware Roofline Model: Upgrading the Loft. *IEEE Comput. Archit. Lett.* **2014**, *13*, 21–24. [\[CrossRef\]](#)
13. Liu, B.; Zou, D.; Feng, L.; Feng, S.; Fu, P.; Li, J. An FPGA-Based CNN Accelerator Integrating Depthwise Separable Convolution. *Electronics* **2019**, *8*, 281. [\[CrossRef\]](#)
14. Huang, H.; Liu, Z.; Chen, T.; Hu, X.; Zhang, Q.; Xiong, X. Design Space Exploration for YOLO Neural Network Accelerator. *Electronics* **2020**, *9*, 1921. [\[CrossRef\]](#)
15. Ding, N.; Awan, M.; Williams, S. Instruction Roofline: An insightful visual performance model for GPUs. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6591. [\[CrossRef\]](#)
16. Panagou, I.-M.; Gkeka, M.-R.; Patras, A.; Lalis, S.; Antonopoulos, C.D.; Bellas, N. FPGA Roofline Modeling and Its Application to Visual SLAM. In *Proceedings of the 32nd International Conference on Field-Programmable Logic and Applications (FPL2020)*, Belfast, UK, 29 August–2 September 2022; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2022; pp. 130–135.
17. Antepara, O.; Oliker, L.; Williams, S. Roofline Analysis of Tightly-Coupled CPU-GPU Superchips: A Study on MI300A and GH200. In *Proceedings of the Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC Workshops '25*, St Louis, MO, USA, 16–21 November 2025; Association for Computing Machinery: New York, NY, USA, 2025; pp. 1205–1216.
18. Marques, D.; Duarte, H.; Ilic, A.; Sousa, L.; Belenov, R.; Thierry, P.; Matveev, Z.A. Performance Analysis with Cache-Aware Roofline Model in Intel Advisor. In *Proceedings of the 2017 International Conference on High Performance Computing & Simulation (HPCS)*, Genoa, Italy, 17–21 July 2017; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2017; pp. 898–907.
19. Wu, S.; Yang, H.; You, X.; Gong, R.; Liu, Y.; Luan, Z.; Qian, D. PProof: A Comprehensive Hierarchical Profiling Framework for Deep Neural Networks with Roofline Analysis. In *Proceedings of the 53rd International Conference on Parallel Processing, ICPP '24*, Gotland, Sweden, 12–15 August 2024; Association for Computing Machinery: New York, NY, USA, 2024; pp. 822–832.
20. Menear, K.; Nag, A.; Perr-Sauer, J.; Lunacek, M.; Potter, K.; Duplyakin, D. Mastering HPC Runtime Prediction: From Observing Patterns to a Methodological Approach. In *Proceedings of the Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good, PEARC '23*, Portland, OR, USA, 23–27 July 2023; Association for Computing Machinery: New York, NY, USA, 2023; pp. 75–85.
21. Chen, X.; Zhang, H.; Bai, H.; Yang, C.; Zhao, X.; Li, B. Runtime Prediction of High-Performance Computing Jobs Based on Ensemble Learning. In *Proceedings of the 4th International Conference on High Performance Compilation, Computing and Communications, HP3C 2020*, Guangzhou, China, 27–29 June 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 56–62.
22. Veigas, K.L.; Chinnici, A.; De Chiara, D.; Chinnici, M. Towards Energy Efficiency of HPC Data Centers: A Data-Driven Analytical Visualization Dashboard Prototype Approach. *Electronics* **2025**, *14*, 3170. [\[CrossRef\]](#)
23. Chinnici, A.; Ahmadzada, E.; Kor, A.-L.; De Chiara, D.; Domínguez-Díaz, A.; de Marcos Ortega, L.; Chinnici, M. Towards Sustainability and Energy Efficiency Using Data Analytics for HPC Data Center. *Electronics* **2024**, *13*, 3542. [\[CrossRef\]](#)
24. Netti, A.; Kiziltan, Z.; Babaoglu, O.; Sirbu, A.; Bartolini, A.; Borghesi, A. A Machine Learning Approach to Online Fault Classification in HPC Systems. *Future Gener. Comput. Syst.* **2020**, *110*, 1009–1022. [\[CrossRef\]](#)
25. Kamila, N.K.; Frnda, J.; Pani, S.K.; Das, R.; Islam, S.M.N.; Bharti, P.K.; Muduli, K. Machine Learning Model Design for High Performance Cloud Computing & Load Balancing Resiliency: An Innovative Approach. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 9991–10009. [\[CrossRef\]](#)
26. Tian, X.; Li, X.; Zhang, J.; Zhao, Z.; Wang, C.; Wang, X.; Wang, J. An Online Incremental Learning Framework for HPC Job Power Consumption Prediction. In *Proceedings of the 2023 7th International Conference on High Performance Compilation, Computing and Communications, HP3C '23*, Jinan, China, 17–19 June 2023; Association for Computing Machinery: New York, NY, USA, 2023; pp. 176–183.
27. Yokelson, D.; Charest, M.R.J.; Li, Y.W. HPC Application Performance Prediction with Machine Learning on New Architectures. In *Proceedings of the 2023 on Performance Engineering, Modelling, Analysis, and Visualization Strategy, PERMAVOST '23*, Orlando, FL, USA, 20–30 June 2023; Association for Computing Machinery: New York, NY, USA, 2023; pp. 1–8.
28. Bolet, G.; Georgakoudis, G.; Menon, H.; Parasyris, K.; Hasabnis, N.; Estes, H.; Cameron, K.; Oren, G. Can Large Language Models Predict Parallel Code Performance? In *Proceedings of the 34th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '25*, Notre Dame, IN, USA, 20–23 July 2025; Association for Computing Machinery: New York, NY, USA, 2025; pp. 1–6.
29. Zou, Y.; Lin, Z.; Zhang, Q.; Liu, Z.; Xu, Z. Large language models enable semantic-guided hierarchical games for intelligent battery coordination. *Adv. Eng. Inform.* **2026**, *71*, 104312. [\[CrossRef\]](#)

30. Antici, F.; Yamamoto, K.; Domke, J.; Kiziltan, Z. Augmenting ML-based Predictive Modelling with NLP to Forecast a Job's Power Consumption. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W '23, Denver, CO, USA, 12–17 November 2023*; Association for Computing Machinery: New York, NY, USA, 2023; pp. 1820–1830.
31. Antici, F.; Seyedkazemi Ardebili, M.; Bartolini, A.; Kiziltan, Z. PM100: A Job Power Consumption Dataset of a Large-scale Production HPC System. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W '23, Denver, CO, USA, 12–17 November 2023*; Association for Computing Machinery: New York, NY, USA, 2023; pp. 1812–1819.
32. Antici, F.; Bartolini, A.; Domke, J.; Kiziltan, Z.; Yamamoto, K. F-DATA: A Fugaku Workload Dataset for Job-centric Predictive Modelling in HPC Systems. *Sci. Data* **2025**, *12*, 1321. [[CrossRef](#)] [[PubMed](#)]
33. Antici, F.; Bartolini, A.; Kiziltan, Z.; Babaoglu, O.; Kodama, Y. MCBound: An Online Framework to Characterize and Classify Memory/Compute-bound HPC Jobs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC24), Atlanta, GA, USA, 17–22 November 2024*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2024; pp. 1–15.
34. del Campo, C.; Sanz, B.; Díaz, J.; Onieva, E. A Fuzzy Logic Ensemble Approach to Concept Drift Detection. In *Proceedings of the Hybrid Artificial Intelligent Systems: 18th International Conference, HAIS 2023, Salamanca, Spain, 5–7 September 2023*; Springer: Cham, Switzerland, 2023; pp. 84–96.
35. Schlimmer, J.C.; Granger, R.H. Incremental Learning from Noisy Data. *Mach. Learn.* **1986**, *1*, 317–354. [[CrossRef](#)]
36. Lin, X.; Chang, L.; Nie, X.; Dong, F. Temporal Attention for Few-Shot Concept Drift Detection in Streaming Data. *Electronics* **2024**, *13*, 2183. [[CrossRef](#)]
37. Bayram, F.; Ahmed, B.S.; Kassler, A. From Concept Drift to Model Degradation: An Overview on Performance-Aware Drift Detectors. *Knowl.-Based Syst.* **2022**, *245*, 108632. [[CrossRef](#)]
38. Gong, Q.; Ma, S.; Zhang, N.; Liu, H.; Gao, H.; Zhao, Y.; Jiang, X.; Tu, W.; Chen, C.; Yang, F. A Semi-Supervised Clustering Algorithm for Underground Disaster Monitoring and Early Warning. *Electronics* **2025**, *14*, 965. [[CrossRef](#)]
39. Beshah, Y.K.; Abebe, S.L.; Melaku, H.M. Drift Adaptive Online DDoS Attack Detection Framework for IoT System. *Electronics* **2024**, *13*, 1004. [[CrossRef](#)]
40. Hu, H.; Kantardzic, M.; Sethi, T.S. No Free Lunch Theorem for Concept Drift Detection in Streaming Data Classification: A Review. *WIREs Data Min. Knowl. Discov.* **2019**, *10*, e1327. [[CrossRef](#)]
41. Şenol, A.; Agrawal, G.; Liu, H. Domain Knowledge-Enhanced LLMs for Fraud and Concept Drift Detection. *Electronics* **2026**, *15*, 534. [[CrossRef](#)]
42. Ghahramanian, P.; Bakhshi, S.; Bonab, H.; Can, F. A Novel Neural Ensemble Architecture for On-the-fly Classification of Evolving Text Streams. *ACM Trans. Knowl. Discov. Data* **2024**, *18*, 1–24. [[CrossRef](#)]
43. Li, J.; Malialis, K.; Panayiotou, C.G.; Polycarpou, M.M. Unsupervised Incremental Learning with Dual Concept Drift Detection for Identifying Anomalous Sequences. In *Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN), Yokohama, Japan, 30 June–5 July 2024*; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2024; pp. 1–8.
44. Greco, S.; Vacchetti, B.; Apiletti, D.; Cerquitelli, T. Unsupervised Concept Drift Detection From Deep Learning Representations in Real-Time. *IEEE Trans. Knowl. Data Eng.* **2025**, *37*, 6232–6245. [[CrossRef](#)]
45. Ali, U.; Mahmood, T. A Novel Framework for Concept Drift Detection Using Autoencoders for Classification Problems in Data Streams. *Int. J. Mach. Learn. Cybern.* **2025**, *16*, 397–418. [[CrossRef](#)]
46. Iman, A.N.; Kamal, I.M.; Kim, D.; Bae, H. Continual Learning Approach With Adaptive Memory Mechanism for Predictive Process Monitoring in Dynamic Business Environments. *IEEE Access* **2025**, *13*, 206767–206783. [[CrossRef](#)]
47. Pérez, J.L.M.; Barros, R.S.M.; Santos, S.G.T.C. Enhancing Semi-Supervised Learning With Concept Drift Detection and Self-Training: A Study on Classifier Diversity and Performance. *IEEE Access* **2025**, *13*, 24681–24697. [[CrossRef](#)]
48. Desale, K.S.; Shinde, S.V. Concept Drift Detection and Adaption Framework Using Optimized Deep Learning and Adaptive Sliding Window Approach. *Expert Syst.* **2023**, *40*, e13394. [[CrossRef](#)]
49. Cai, X.; Li, D.; Zou, Y.; Liu, Z.; Heidari, A.A.; Chen, H. A hybrid wind speed forecasting model with rolling mapping decomposition and temporal convolutional networks. *Energy* **2025**, *324*, 135673. [[CrossRef](#)]
50. Suryawanshi, S.; Goswami, A.; Patil, P. Modeling self-supervised learning for handling the concept drift with Ternary-Adaptive Ensemble. *Appl. Soft Comput.* **2025**, *184*, 113772. [[CrossRef](#)]
51. Kidane, L.; Townend, P.; Metsch, T.; Elmroth, E. Automated Hyperparameter Tuning for Adaptive Cloud Workload Prediction. In *Proceedings of the 16th IEEE/ACM International Conference on Utility and Cloud Computing, UCC '23, Taormina, Messina, Italy, 4–7 December 2023*; Association for Computing Machinery: New York, NY, USA, 2023; pp. 1–8.
52. Chen, J.; Yang, S.; Gao, T.; Ying, Y.; Li, T.; Li, P. Multi-type Concept Drift Detection Under a Dual-layer Variable Siding Window in Frequent Pattern Mining with Cloud Computing. *J. Cloud Comput.* **2024**, *13*, 40. [[CrossRef](#)]

53. Xavier, B.M.; Martinello, M.; Trois, C.; Alenca, B.M.; Rios, R.A. Fast Learning Enabled by In-Network Drift Detection. In *Proceedings of the 8th Asia-Pacific Workshop on Networking, APNet '24, Sydney, Australia, 3–4 August 2024*; Association for Computing Machinery: New York, NY, USA, 2024; pp. 129–134.
54. Ross, G.J.; Adams, N.M.; Tasoulis, D.K.; Hand, D.J. Exponentially Weighted Moving Average Charts for Detecting Concept Drift. *Pattern Recognit. Lett.* **2012**, *33*, 191–198. [[CrossRef](#)]
55. Sentence-Transformers. All-MiniLM-L6-v2. Available online: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2> (accessed on 6 January 2026).
56. Xiao, S.; Liu, Z.; Zhang, P.; Muennighoff, N.; Lian, D.; Nie, J.-Y. C-Pack: Packed Resources For General Chinese Embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24, Washington, DC, USA, 14–18 July 2024*; Association for Computing Machinery: New York, NY, USA, 2024; pp. 641–649.
57. Wang, L.; Nan, Y.; Huang, X.; Jiao, B.; Yang, L.; Jiang, D.; Majumder, R.; Wei, F. Text Embeddings by Weakly-Supervised Contrastive Pre-training. *arXiv* **2022**. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.