ORA: Job Runtime Prediction for High-Performance Computing Platforms Using the Online Retrieval-Augmented Language Model

Hongyi Liu
Peking University
Beijing, China
hongyiliu@pku.edu.cn

Yinping Ma*
Peking University
Beijing, China
mayinping@pku.edu.cn

Xiaosong Huang
Peking University
Beijing, China
hxs@stu.pku.edu.cn

Lingzhe Zhang
Peking University
Beijing, China
zhang.lingzhe@stu.pku.edu.cn

Tong Jia
Peking University
Beijing, China
National Key Laboratory of Data Space Technology and System
Beijing, China
jia.tong@pku.edu.cn

Ying Li*
Peking University
Beijing, China
li.ying@pku.edu.cn

Abstract

Accurate job runtime prediction is critical for efficient scheduling in high-performance computing (HPC) platforms. For instance, precise predictions enable techniques such as backfilling, where small jobs are executed ahead of schedule to maximize resource utilization and enhance computational efficiency. However, existing runtime prediction methods primarily rely on job metadata (e.g., submission time, requested runtime, and required memory) while ignoring the content of job scripts, which limits their accuracy. To address this issue, we propose an Online Retrieval-Augmented Language Model (ORA) for job runtime prediction. ORA encodes both metadata and script information from historical jobs into feature vectors to form a database, enabling similarity-based retrieval to assist in predicting the runtime of new jobs. To address distribution shifts, ORA incrementally updates the database without requiring model retraining. Additionally, personalized retrieval mechanisms are employed to mitigate the impact of distribution shifts. To reduce interference caused by repetitive content in retrieved jobs and enhance the learning of essential differences, we design a diff-based contextual learning mechanism. This highlights the differences between the current job and the retrieved jobs, improving the model's ability to capture distinctive features. Experimental results demonstrate that the proposed method outperforms existing baselines, achieving an average accuracy improvement of over 40% in real-world scenarios where the training and testing job times do not overlap, and the metadata does not include running information such as actual memory usage. Ablation studies further highlight the contribution of each component of the proposed method.

CCS Concepts

• **Computing methodologies** → **Temporal reasoning**; Planning with abstraction and generalization.

Keywords

Runtime Prediction, LLMs, Retrieval-Augmented, Online Learning

ACM Reference Format:
Hongyi Liu, Yinping Ma, Xiaosong Huang, Lingzhe Zhang, Tong Jia, and Ying Li. 2025. ORA: Job Runtime Prediction for High-Performance Computing Platforms Using the Online Retrieval-Augmented Language Model. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3721145.3725757

*Co-corresponding author.

ICS '25, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1537-2/25/06
https://doi.org/10.1145/3721145.3725757

1 Introduction

With the increasing computational demands of modern scientific computing and engineering applications, numerous universities [4], institutions [1], and enterprises [2] have established their own high-performance computing (HPC) platforms. These platforms integrate underlying computational resources and provide computing services. Users can request resources such as memory, processors, and runtime, and submit computational scripts (e.g., bash scripts) to form jobs. To efficiently utilize resources and enhance user experience, HPC platforms commonly adopt a scheduling strategy combining first-come-first-serve (FCFS) and backfilling [21]. In FCFS, jobs are executed in the order they are submitted. Backfilling allows certain small jobs to be run out of order when resources are available, provided that this does not compromise the fairness of the FCFS policy. A critical aspect of backfilling is predicting job runtime since users often provide inaccurate estimates [11, 13, 20]. Accurate runtime predictions help ensure that out-of-order execution does not disrupt the overall fairness of the scheduling process.

Existing approaches to job runtime prediction primarily rely on job metadata (e.g., submission time, requested runtime, and memory requirements). These methods leverage statistical learning [7, 9, 10, 17, 18] or deep learning techniques [6, 8] to analyze historical data. However, they often neglect the discriminative information contained in job scripts, which limits prediction accuracy. As illustrated in Figure 1, we compared the discriminative power of static features (e.g., those shown in Table 1 excluding unique identifiers like *job_id* and *submit_time*), running features (e.g., Table 1 excluding the *run_time* target), and script features. The results show that static features provide limited discrimination, with a duplication ratio exceeding 80% across datasets. Incorporating running features significantly enhances job discrimination, especially for the KTH and SDSC datasets. However, since runtime predictions generally occur before job execution, running features (e.g., wait time, memory usage) are often unavailable in real-world scenarios. Excitingly, script features (available in D1 and D2 datasets) also significantly improve discrimination and reduce the duplication ratio. Since scripts are accessible at submission time, they should be utilized for runtime prediction. Yet, research exploring this direction remains limited.

To harness the rich discriminative information in job scripts, we propose leveraging large language models (LLMs) for processing script textual data. Research in natural language [16] and code processing [26] has demonstrated that larger model sizes often yield better performance. Considering that runtime predictions occur during the job queuing phase, the computational overhead of LLMs is acceptable. To address challenges like limited fine-tuning datasets, high computational costs, and scalability issues, we propose using retrieval-augmented generation (RAG) to enhance runtime

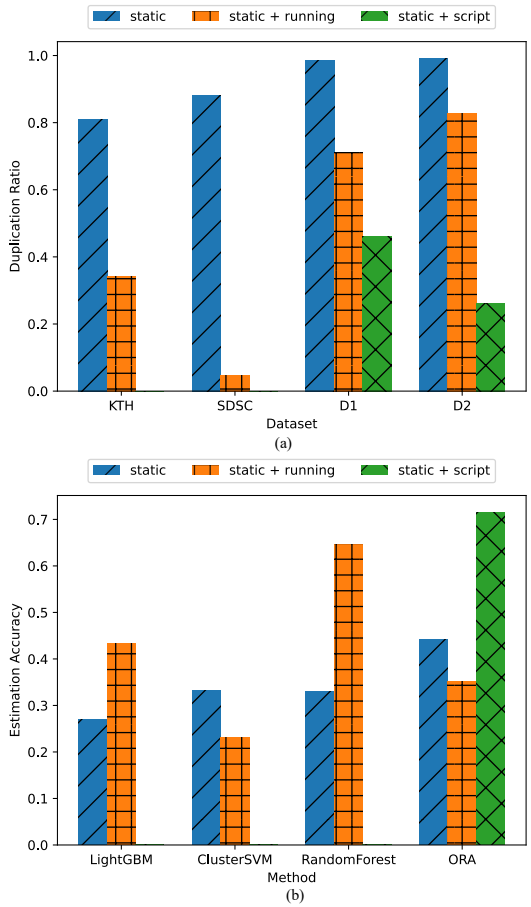

Figure 1: Motivational example. (a) Visualizes the discriminative power of static features, running features, and script features (as defined in Table 1) for job identification. The static features, static + running features, and static + script features are concatenated into strings, which are used as dictionary keys, with the job's *run_time* as the value. The duplication ratio is calculated as $1-(key\#/job\#)$. (b) Visualizes the impact of static features, running features, and script features on the estimation accuracy (EA) of the proposed method (ORA) and baseline methods (LightGBM, ClusterSVM, RandomForest) in the D1 dataset (as described in section 4.1.1). The results indicate that script features provide significant discriminative power for jobs, and incorporating this information notably improves the estimation accuracy of the proposed method.

prediction accuracy. Prior studies [5] have shown that well-designed RAG strategies can achieve performance comparable to fine-tuning.

However, utilizing retrieval-augmented LLMs (RALMs) for job runtime prediction presents two challenges: (1) **Distribution Shifts**: A static vector database struggles to maintain relevance as job distributions change over time, limiting prediction accuracy. As shown in Figure 2, the correlation between jobs diminishes as submission intervals (in terms of time or ID) increase. (2) **Redundancy in retrieved samples**: High redundancy in historical samples restricts LLMs from learning essential information, further constraining prediction accuracy. Figure 3 highlights how users may submit numerous similar jobs (e.g., due to debugging or parameter tuning), where meaningful distinctions are lost amidst repetitive contextual information.

To address these challenges, we propose an Online Retrieval Augmented (ORA) language model for job runtime prediction, as illustrated in Figure 4. ORA encodes historical job metadata and script information into feature vectors to construct a database, enabling similarity-based retrieval for runtime prediction. To tackle distribution shifts, the database is continuously updated without retraining the model, and personalized retrieval strategies are employed. To mitigate the impact of redundant information from historical samples, we design a diff-based context learning approach that highlights distinctions between the current job and retrieved samples, enabling LLMs to focus on critical differences. Experimental results demonstrate that ORA surpasses existing baselines, improving prediction accuracy by over 40% on average in real-world scenarios where training and testing jobs are temporally disjoint, and runtime metadata excludes running features like memory usage. Ablation studies further validate the contributions of individual components in the proposed approach.

In summary, this paper makes the following contributions:

- Introduces script content into runtime prediction and, for the first time, applies LLMs to this task, proposing an Online Retrieval-Augmented Language Model.
- Designs practical techniques to address challenges in applying RALM to runtime prediction, including continuous database updates, personalized retrieval strategies to mitigate distribution shifts, and a diff-based approach to mitigate redundancy in retrieved samples.
- Demonstrates significant performance gains through extensive experiments and validates the contributions of individual components via ablation studies. To facilitate reproducibility, we have made our code publicly available [3].

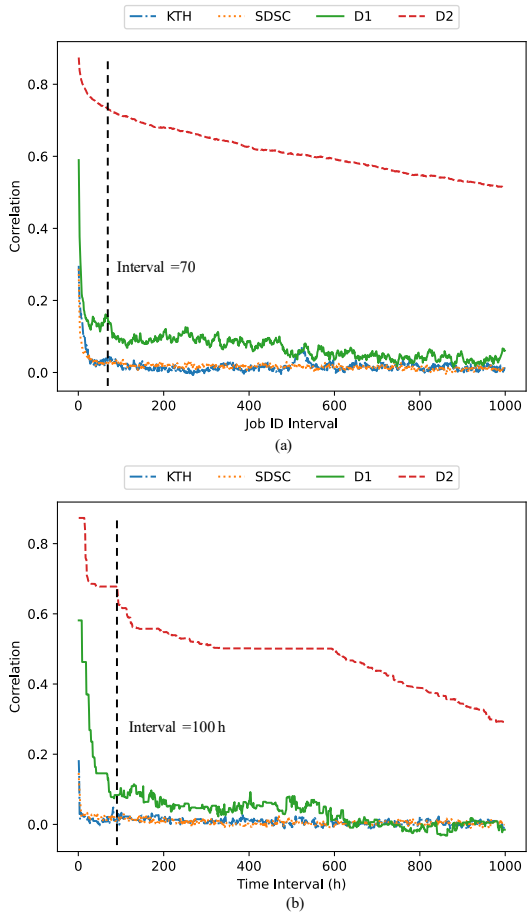

Figure 2: Impact of the interval (time or ID) between submitted jobs on the correlation between jobs. (a) Visualizes the change in the Pearson correlation coefficient of job runtime as the interval between job IDs increases. (b) Visualizes the change in the Pearson correlation coefficient of job runtime as the time interval between job submissions increases. The results show that as the interval between jobs increases, the correlation in runtime significantly decreases, indicating that the distribution of jobs is undergoing continuous evolution.

2 Related Work

2.1 Runtime Prediction

Existing methods for job runtime prediction can be categorized into two primary approaches based on the models employed: statistical learning-based methods and deep learning-based methods. *Statistical Learning-Based Methods.* Rauschmayer [18] utilized linear regression and maximum likelihood estimation to predict job runtime. Renato et al. [9]

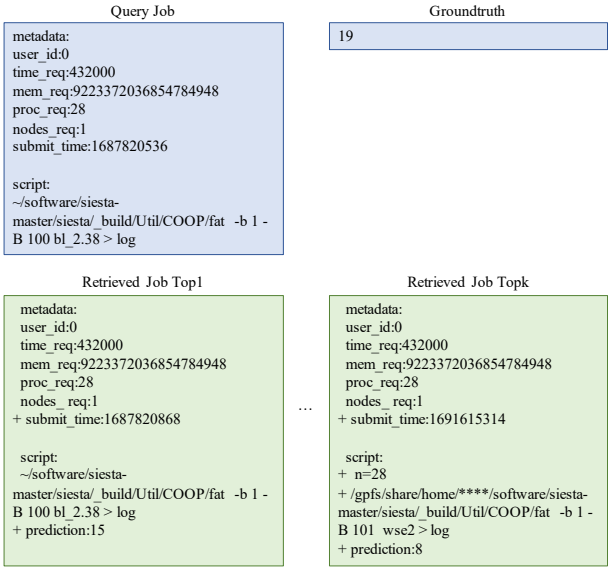

Figure 3: The examples of query jobs and retrieved jobs.

adopted the k-nearest neighbor (KNN) algorithm to model historical data and predict job runtime. Due to the heterogeneous characteristics of HPC systems, a single machine learning model often fails to perform consistently, prompting the widespread adoption of ensemble learning techniques for runtime prediction. For instance, Park et al. [17] identified runtime-relevant features, partitioned historical data into clusters, and constructed support vector machine (SVM) models for each cluster. Similarly, Dai et al. [10] clustered historical data and built SVM models for each cluster while introducing a mechanism for continuously updating the models to enhance adaptability to new data. Chen et al. [7] compared various models, including both single and ensemble models, for runtime prediction tasks on HPC systems. They recommended LightGBM as a computationally efficient model with high accuracy, comparable to RandomForest in terms of performance. *Deep Learning-Based Method*s. Cheon et al. [8] utilized deep neural networks (DNNs) to automatically extract features and predict job runtimes. Chen et al. [6] introduced temporal dependencies between jobs, formulating runtime prediction as a time-series forecasting task. They employed Transformer models to capture inter-job temporal relationships, improving the accuracy of runtime predictions. While the aforementioned methods have achieved notable results, many rely on running features that are unavailable at the time of prediction, limiting their applicability in real-world HPC scenarios. Furthermore, these approaches overlook the rich, discriminative textual features embedded in job scripts, which constrains their performance in realistic job runtime prediction tasks.

2.2 Retrieval-augmented generation

Retrieval-augmented generation (RAG) is a technique that enhances a model's generation capabilities by incorporating additional knowledge retrieved from external sources. Due to its interpretability, scalability, and adaptability, this paradigm has been widely applied in knowledge-intensive generation tasks, including code generation [26], dialogue generation [16], and machine translation [22]. In the RAG workflow, a retriever fetches supplementary knowledge from an external knowledge base, which is then combined with the parametric knowledge learned during pretraining by the generator to solve the given task. Retrievers typically use either sparse or dense similarity measures to filter relevant information. Sparse vector retrieval methods, such as TF-IDF and BM25 [19], compute similarity by matching keywords in a sparse bag-of-words representation space. In contrast, dense vector retrieval relies on the inner product of low-dimensional dense vectors obtained from neural networks [15]. Sparse retrieval excels at identifying exact term overlaps, while dense retrieval captures semantically related but lexically divergent information. Both approaches fundamentally rely on similarity-based retrieval mechanisms. In the NLP domain, a few studies [23–25] have explored scenarios where retrieved content contains noise. However, these studies primarily focus on irrelevant knowledge retrieved due to imperfections in retrieval algorithms. In contrast, the challenge discussed in this work addresses the issue of excessive redundant and repetitive information within the retrieved content. It is rarely explored. Such redundancy limits the contextual learning capabilities of large language models (LLMs), thereby constraining their performance in tasks like job runtime prediction.

3 Method

In this section, we first present the formal definition of the job runtime prediction problem. Then, we detail the implementation of each module of the proposed method, including the offline database construction module and the online prediction module. Furthermore, within the online prediction module, we elaborate on two key components: retrieval-augmented prediction and online database updating.

3.1 Problem Statement

The objective of job runtime prediction is to estimate the runtime of a given job x_i at the time of its submission. Each job x_i is characterized by numerical metadata x_i^m and textual script information x_i^s. The goal is to predict the runtime $\hat{y}_i =$

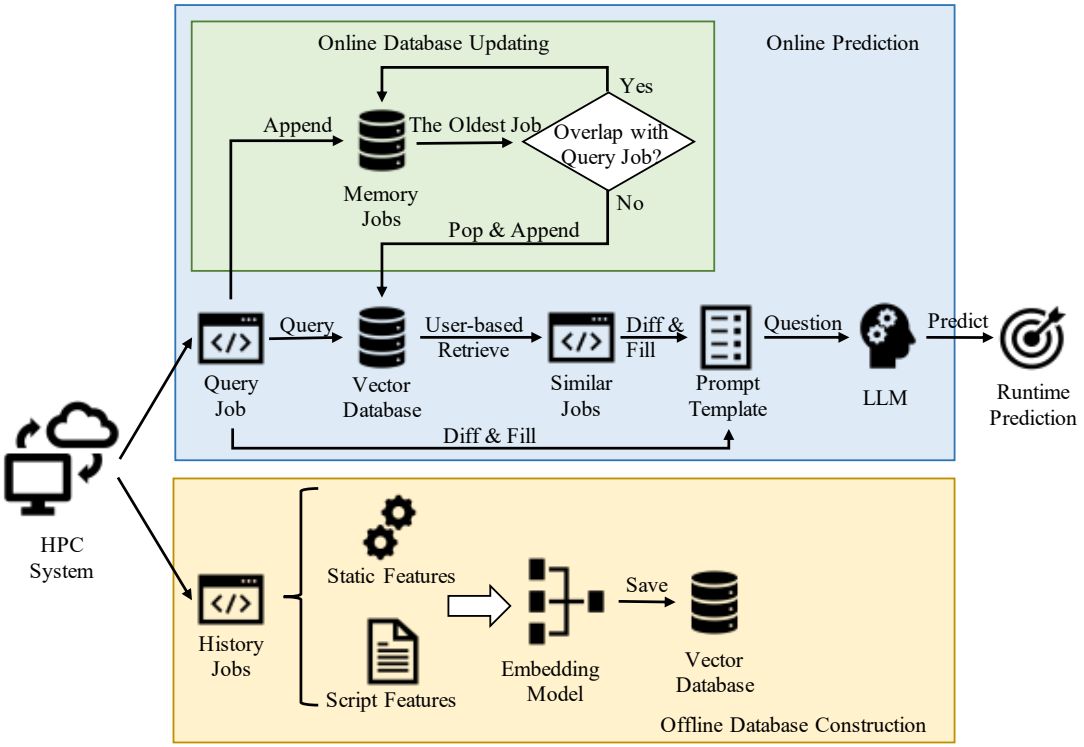

Figure 4: The overview of the proposed method.

$f(x_i)$ such that the predicted runtime $\hat{y}_i$ closely approximates the actual runtime y_i.

3.2 Offline Database Construction

To build a vector database from historical jobs in an HPC system, job information is first converted into textual representations. These representations are then processed using a text embedder to generate job vectors, which are stored in the database. For the j-th metadata field x_i^j of the i-th job, the value is converted into a string of the format "$name^j : x_i^j$", where the $name^j$ is the field name of the j-th metadata field. Similarly, the script information x_i^s is converted into the format "$script : x_i^s$", where the $script^j$ is the plain text, with all textual contents concatenated to form the complete textual representation of the job, as shown in the Query Job example in Figure 3. The resulting job vectors are stored in the vector database.

Each stored sample is configured with additional attributes, including:

- *user_id*: Used for personalized retrieval.
- *submit_time* and *end_time*: Used for online database updates.
- *run_time*: Used as the label for historical jobs.

Notably, the textual representation stored in the vector database excludes the *run_time* attribute. This omission ensures consistency between the representation of new *Query Jobs* and the stored *History Jobs*, as the runtime is unavailable for *Query Jobs* during prediction. Maintaining identical construction methods for *Query Jobs* and *History Jobs* prevents structural bias in the representations.

3.3 Online Prediction

When a new job arrives in the HPC system, it is first converted into a textual representation (*Query Job*) following the method described in the previous section. This representation is then processed through two main stages: retrieval-augmented prediction and online database updating.

User-based Retrieve. To minimize the distribution gap between historical and future jobs, we leverage the *user_id* attribute to filter the jobs in the database. Only jobs submitted by the same user as the *Query Job* are retained for retrieval, ensuring the retrieved jobs are more relevant to the *Query Job*.

Diff-based In-context Learning. To address the issue of high redundancy in retrieved historical samples that can hinder the large language model's (LLM) ability to extract meaningful information, we apply a 'diff' operation. This operation highlights the content added in the retrieved job

```
"""
You are an expert in job runtime duration prediction.
There are some retrieved history jobs that similar to the job
waitting to predict.
They are displaied through a diff format.
{context}

Please predict the job runtime duration based on its
matedata, script, and retrieved jobs.
The matedata and script of the job waitting to predict is:
{question}

Your output should only include the runtime, e.g. 10 s.
It means that the script is likely to run for 10 seconds.
Note: DO NOT OUTPUT ANYTHING OTHER THAN
THE RUNTIME.
"""
```

Figure 5: Prompt template.

compared to the *Query Job* by marking it with a "+" symbol. Notably, content reductions in the retrieved job are not marked to minimize the number of tokens fed into the LLM. Additionally, unaltered content is retained, as it contributes to the LLM's understanding of the mapping between similar job types and their runtimes. Removing such invariant content would negatively impact prediction accuracy.

Prompt Construction. The processed content from the diff operation, along with the corresponding runtime prediction, is structured as shown in Figure 3 (e.g., *Retrieved Job Top1*). The contents of the top n_k retrieved jobs are concatenated to form the *context*, while the textual content of the *Query Job* is used as the *question*. These components are then organized into the template illustrated in Figure 5, forming the input to the LLM. The LLM outputs the predicted runtime for the *Query Job*.

3.3.1 Online Database Updating. To continually reduce the distribution gap between historical and future jobs, the vector database is updated dynamically. In real-world scenarios, only completed jobs can be added to the vector database. To facilitate this, a memory queue is maintained to store newly submitted jobs. Once the job at the head of the queue is completed (i.e., its *end_time* is less than the *submit_time* of the *Query Job*), it is removed from the queue and added to the vector database. This ensures that the jobs in the vector database do not overlap in time with the submission of the *Query Job*, maintaining consistency with the real-world scenarios.

4 Experiment

In this section, we first describe the experimental setup, including datasets, baseline methods, evaluation metrics, and implementation details. Subsequently, we address the following research questions (RQs):

- RQ1: How effective is the proposed method overall?
- RQ2: Are the individual components of the proposed method effective?
- RQ3: Is the proposed method sensitive to parameter variations?

4.1 Experiment Setup

4.1.1 Datasets. To validate the effectiveness of the proposed method, we conducted experiments on four datasets, as summarized in Table 1. These include two open-source datasets (KTH, SDSC) and two proprietary industrial datasets (D1, D2).

- KTH Dataset: The KTH dataset [12] contains 11 months of workload data collected from the 100-node IBM SP2 system at the Swedish Royal Institute of Technology (KTH) in Stockholm.
- SDSC Dataset: The SDSC dataset [12] comprises 2 years of workload data from the San Diego Supercomputer Center. It includes information on the user, account, and application, as well as requested and used nodes and time, CPU time, and submit, wait, and run times.
- D1 Dataset: The D1 dataset contains 1 year (January 2023 to January 2024) of workload data collected from a university's high-performance computing (HPC) platform. This dataset is designed for general-purpose computing tasks across the university and includes both job metadata and script information.
- D2 Dataset: Similar to the D1 dataset, the D2 dataset focuses primarily on life sciences research computing tasks.

4.1.2 Baselines. We compared the proposed method with several baseline approaches, spanning statistical learning-based methods (LightGBM, ClusterSVM, RandomForest) and deep learning-based methods (DNN, Transformer).

- LightGBM [14]: An advanced decision-tree-based ensemble regression method that offers higher computational efficiency compared to RandomForest. LightGBM is also the runtime prediction algorithm currently deployed in the production environment of the university's high-performance computing platform where the D1 and D2 datasets originate.
- RandomForest [7]: A standard decision-tree-based ensemble method, implemented using the sklearn library.
- ClusterSVM [10]: This method employs clustering to partition jobs into groups. An individual SVM model is trained for each cluster to predict job runtimes. To

Table 1: Dataset Overview

Dataset	KTH	SDSC	D1	D2
Job #	28,476	58,715	19,564	187,231
Feature #	18	18	11	11
Static Feature	job_id, submit_time, user_id, group_id, time_req, mem_req, proc_req, exe_num, q_num, partition_num, prev_job, think_time	Same as KTH	job_id, submit_time, user_id, group_id, time_req, mem_req, proc_req, nodes_req	Same as D1
Running Feature	wait_time, proc_used, cputime_used, mem_used, status, run_time	Same as KTH	start_time, run_time	Same as D1
Script Feature	-	-	bash_script	Same as D1

address distribution shifts, the models are updated every 700 jobs.

- DNN [8]: A runtime prediction approach leveraging deep neural networks (DNN).
- Transformer [6]: This method clusters user IDs to map them to user categories, improving prediction accuracy across different user groups. It reformulates the job runtime prediction task as a time-series forecasting problem, leveraging Transformers to capture sequential dependencies between jobs and enhance runtime prediction.

4.1.3 Evaluation Metrics. To evaluate the accuracy of job runtime prediction, we utilized standard metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and Estimation Accuracy (EA). The Estimation Accuracy (EA) metric is widely adopted for assessing the precision of job runtime predictions, as referenced in [6, 10]. The following formulas define the estimation accuracy for an individual job (ea_i) and the average estimation accuracy across all jobs (EA):

$$ea_i = \begin{cases} \frac{f(x_i)}{y_i}, f(x_i) \leq y_i \\ \frac{y_i}{f(x_i)}, f(x_i) > y_i \end{cases} \quad (1)$$

$$EA = 1/n * \sum_{i=1}^{n} ea_i. \quad (2)$$

Here, n represents the total number of jobs, $f(x_i)$ denotes the predicted runtime of the i-th job, and y_i indicates its actual runtime.

4.1.4 Implementation Details. For all datasets, we used the following consistent hyperparameters: the number of retrieved historical samples $n_h = 7$, and the training-to-testing split ratio of 9:1, where training data is used to construct the vector database or model. The LLM used was Qwen2.5:14b. To better reflect real-world scenarios, only static features and script features from each dataset were utilized for job time prediction. Ollama was used to build local LLMs, Chroma was used to construct the vector database, and LangChain was employed to build the pipeline.

4.2 RQ1: Performance Study

To evaluate the effectiveness of the proposed method, we compared it against multiple baseline approaches, as shown in Table 2. The results demonstrate that the proposed method consistently outperforms the baselines across four datasets, achieving significantly lower MAE and MSE and substantially higher EA. On average, the proposed method improves EA by 43% across the four datasets, validating its effectiveness. One primary reason for the poorer performance of baseline methods is their reliance solely on static features, without incorporating running features. As illustrated in Figure 1(b), when running features are added to the RandomForest method, its EA improves significantly. However, it still falls short of the proposed method, further corroborating the efficacy of the proposed approach.

When comparing the proposed method with existing job runtime prediction techniques, it exhibits distinct advantages over statistical learning-based approaches (e.g., LightGBM, ClusterSVM, RandomForest). These improvements stem from incorporating more discriminative script information and leveraging deep learning for automatic feature extraction. Compared to deep learning-based methods (e.g., DNN, Transformer), the proposed method integrates script information and uses language models to capture script semantics. Additionally, employing a retrieval-augmented generation (RAG) approach enhances runtime prediction accuracy.

Table 2: Main Result. *MAE* and *MSE* should be minimized, while *EA* should be maximized. The best values are highlighted for easy reference.

	KTH			SDSC			D1			D2			Average
	MAE	MSE	EA	MAE	MSE	EA	MAE	MSE	EA	MAE	MSE	EA	EA
LightGBM (NeurIPS 17)	0.0367	0.0057	0.4235	0.0134	0.0006	0.3547	0.1044	0.0284	0.2691	0.061	0.0123	0.3701	0.3543
ClusterSVM (SC 22)	0.0379	0.0052	0.4166	0.0151	0.0006	0.2961	0.1056	0.0268	0.3331	0.0659	0.0115	0.2849	0.3326
RandomForest (HP3C 20)	0.0367	0.0054	0.4584	0.0139	0.0006	0.3577	0.1116	0.0288	0.3309	0.1459	0.0428	0.303	0.3625
DNN (Clust Comput 21)	0.0338	0.0056	0.5164	0.0145	0.0008	0.3125	0.1064	0.0354	0.2156	0.0523	0.0141	0.3249	0.3423
Transformer (J Supercomput 23)	0.0342	0.0053	0.5069	0.0145	0.0007	0.3108	0.1082	0.0363	0.1998	0.0519	0.0139	0.3139	0.3328
ORA (*Our*)	**0.0198**	**0.0039**	**0.8164**	**0.0052**	**0.0003**	**0.7825**	**0.0624**	**0.0239**	**0.7402**	**0.0301**	**0.0102**	**0.7189**	**0.7645**

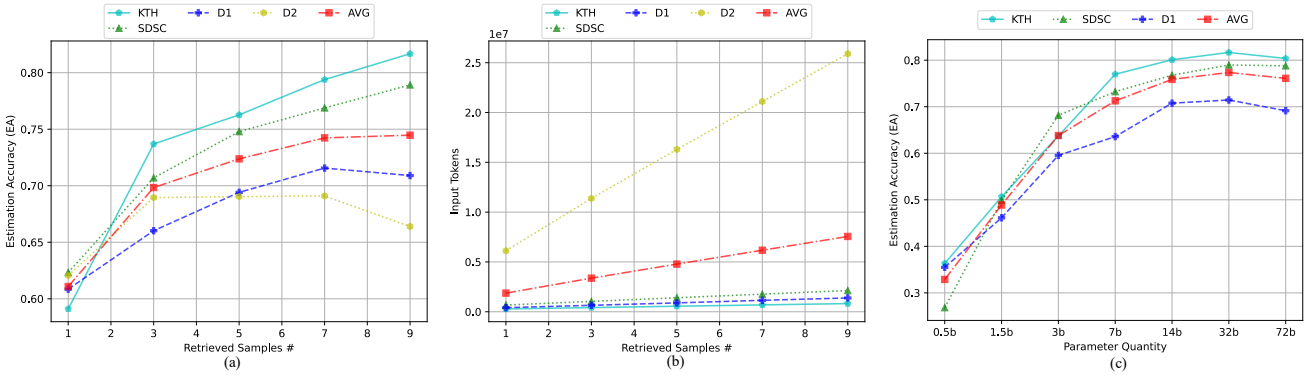

Figure 6: Parameter sensitive.

Table 3: Ablation Study on Modules.

EA	KTH	SDSC	D1	D2	Average
ORA (*Our*)	**0.8164**	**0.7825**	**0.7402**	**0.7189**	**0.7645**
wo Update	0.7938	0.7688	0.7156	0.6909	0.7422
wo User	0.7737	0.7401	0.6845	0.7041	0.7256
wo Diff	0.8012	0.7826	0.7036	0.7139	0.7503
wo RAG	0.3255	0.2636	0.1525	0.2686	0.2525
wo LLM	0.5244	0.5298	0.5201	0.6326	0.5517
w BERT	0.5164	0.3125	0.2854	0.3549	0.3673

4.3 RQ2: Ablation Study

To validate the effectiveness of each component of the proposed method, we conducted a comprehensive ablation study, as shown in Table 3.

Effectiveness of RAG: In Table 3, *wo RAG* represents a variant of the proposed method, where the LLM input includes only the information of the Query Job, without incorporating similar historical jobs. This represents a zero-shot scenario for job runtime prediction. The results show that compared to *wo RAG*, the proposed method *ORA* significantly improves, validating the effectiveness of RAG and confirming that historical similar jobs indeed contribute to better job runtime prediction for the Query Job.

Effectiveness of User Personalization in Retrieval: In Table 3, *wo User* represents a variant of the proposed method where historical jobs are retrieved without considering that the user of the historical jobs must match the user of the Query Job. This represents a general RAG approach. The results indicate that *ORA*, the proposed method, outperforms *wo User*, confirming the effectiveness of user personalization in retrieval. This shows that historical jobs from the same user are more relevant and help mitigate data distribution shifts to some extent.

Effectiveness of Diff-based In-context Learning: In Table 3, *wo Diff* represents a variant of the proposed method where the retrieved historical jobs do not use the Diff operation to mark differences from the Query Job. The results demonstrate that *ORA* outperforms *wo Diff*, validating the effectiveness of Diff-based in-context learning. This suggests that redundant information in the retrieved historical jobs can hinder LLM's context learning and affect the accuracy of job runtime predictions. By highlighting the differences, LLM can better understand the inherent distinctions between different samples, leading to improved prediction accuracy.

Table 4: Ablation Study on Base Model.

EA	KTH	SDSC	Average	D1	D2	Average
Qwen2.5:14b	**0.8164**	**0.7825**	**0.7994**	**0.7402**	**0.7189**	**0.7295**
Phi3:14b	0.6658	0.5199	0.5928	0.2236	0.6048	0.4142
Llama3.2:3b	0.3321	0.3139	0.3231	0.3639	0.3412	0.3525
GPT-4o	0.8038	0.7807	0.7922	-	-	-

Table 5: Efficiency Study.

Time (s)	KTH	SDSC	D1	D2	Average
LightGBM	1.27	2.11	1.21	4.71	2.32
ClusterSVM	0.55	0.95	1.41	2.88	1.44
RandomForest	11.07	21.73	5.14	43.49	20.35
DNN	1.41	2.97	0.98	9.32	3.67
Transformer	2.91	5.98	2.01	18.98	7.47
ORA (*Our*)	699.92	1865.37	894.38	12173.27	3908.23

Effectiveness of Online Database Update: In Table 3, *wo Update* represents a variant of the proposed method where the vector database is not updated online. The results show that *ORA* outperforms *wo Update*, demonstrating the effectiveness of online database updates. The reason behind this is that continuously updating the database helps mitigate data distribution shifts over time.

Effectiveness of LLM: In Table 3, *wo LLM* represents a variant of the proposed method where job runtime predictions are directly based on the average of the retrieved historical jobs. *w BERT* represents a variant of the DNN method [8], which uses BERT to encode job scripts into feature vectors, along with other job metadata, as inputs to a DNN model trained on the training set and tested on the test set. The results show that *ORA* significantly outperforms *wo LLM*, validating the effectiveness of using LLM to analyze historical jobs. LLM can leverage the historical jobs to predict the Query Job's runtime more effectively. The results also show that while *w BERT* achieves better prediction performance compared to the DNN method in Table 2, due to the incorporation of job script information, it still lags behind *ORA*. This is because a single DNN model is not sufficient to capture all the relevant features of job runtime, while LLM has stronger world knowledge and analytical capabilities, resulting in better prediction accuracy.

4.4 RQ3: Parameter Sensitivity

To evaluate the parameter sensitivity of the proposed method, we examined the impact of several hyperparameters unique to our method on the EA value under different settings, including the number of historical samples retrieved (n_h), the choice of LLM model, and the quantity of parameters in the LLM.

Number of Retrieved Historical Samples (n_h): We systematically varied $n_h \in \{1, 3, 5, 7, 9\}$ and analyzed its impact on the EA value, as shown in Figure 6(a). The results indicate that as n_h increases, the EA value initially increases and then plateaus. This is because, at first, increasing the number of retrieved samples provides more information for the LLM, thereby improving the accuracy of job runtime predictions. However, as the number of samples grows too large, the amount of useful information starts to diminish, and irrelevant or low-correlated data may interfere, leading to a decrease in prediction accuracy. We also visualized the number of tokens consumed for different values of n_h, as shown in Figure 6(b). The results clearly show a linear increase in token consumption with the growth of n_h. Therefore, to balance token consumption and prediction accuracy, we set $n_h = 7$.

LLM Model Selection: We explored the impact of different LLM models on the performance of the proposed method, as shown in Table 4. We tested three open-source models: Qwen2.5:14b, Phi3:14b, and Llama3.2:3b, along with a closed-source model, GPT4-o (which was only evaluated on open-source datasets to avoid privacy issues). The results demonstrate that the choice of LLM model significantly affects the performance of the proposed method. Specifically, as the LLM model's capability increases, the job runtime prediction accuracy improves. Notably, Qwen2.5:14b performed even slightly better than GPT4-o, which provides hope for deploying efficient models with fewer parameters in local environments.

LLM Parameter Size: To further investigate the impact of LLM parameter size on the proposed method, we conducted a parameter sensitivity test using models from the Qwen2.5 series, as shown in Figure 6(c). The results show that as the number of parameters increases, the EA value rises initially and then decreases. This is because, at first, the increasing model size enhances the LLM's understanding ability, thus improving job runtime prediction accuracy. However, when the model size becomes too large, the prediction task differs from a generation task, and excessive model complexity may limit prediction accuracy. To balance prediction performance and computational cost, we selected Qwen2.5:14b as the base LLM for our method.

5 Discussion

5.1 Limitations and Future Work

One limitation of the proposed method is its relatively low time efficiency. To evaluate the time efficiency, we compared the total training and testing time of the proposed method with the baseline methods across all datasets, as shown in Table 5. The results indicate that the proposed method, *ORA*, does indeed suffer from time efficiency drawbacks. However,

considering the significant improvement in accuracy and the fact that each submitted job only needs to be predicted once, which helps the backfilling algorithm better schedule jobs, the time overhead is acceptable. In future work, we plan to design more efficient job runtime prediction methods.

5.2 Threats to Validity

An external validity threat is that only the D1 and D2 datasets contain job script information, and both datasets come from the same university's high-performance computing platform, which could affect the generalizability of the proposed method. However, D1 and D2 are actually sourced from two distinct computing clusters, which we believe mitigates this concern to some extent. Additionally, due to privacy restrictions, there is currently a lack of open-source datasets containing job script information. In the future, we plan to consider removing sensitive information and releasing an open-source dataset to promote further research in this area. We also intend to validate the proposed method on more high-performance computing platforms.

6 Conclusion

Existing job runtime prediction methods primarily rely on job metadata (such as submission time, required runtime, and memory size), while overlooking the job script content, which limits prediction accuracy. To address this issue, we propose an online retrieval-augmented language model (ORA) for job runtime prediction. The model encodes both historical job metadata and script information into feature vectors to form a database, from which similar jobs are retrieved to aid in predicting the runtime of new jobs. Additionally, to mitigate distributional shifts, the database is continuously updated without the need to retrain the model. We also incorporate user-personalized retrieval to further alleviate distributional variations. To address the problem of redundant content in retrieved jobs interfering with the LLM's ability to learn meaningful knowledge, we design a diff-based contextual learning method to highlight parts of the job that differ from the current job, thereby enhancing the LLM's learning of the varying components. Experiments demonstrate that the proposed method outperforms existing baseline methods, achieving an average accuracy improvement of over 40% in real-world scenarios. Ablation studies confirm the contribution of each component of the proposed method.

Acknowledgments

This work was supported by Key R&D Project of Guangdong Province (No.2020B010164003). This work was also supported by Special Funds for Construction of Innovative Provinces in Hunan Province under contract 2023GK1010 and High Performance Computing Platform of Peking University.

References

[1] 2025. National Partnership for Advanced Computational Infrastructure. https://www.nsf.gov/awardsearch/showAward?AWD_ID=9619020&HistoricalAwards=false.
[2] 2025. National Partnership for Advanced Computational Infrastructure. https://aliyun.com.
[3] 2025. ORA. https://github.com/lhysgithub/ORA.
[4] 2025. PDC Center for High Performance Computing. https://www.pdc.kth.se.
[5] Angels Balaguer, Vinamra Benara, Renato Luiz de Freitas Cunha, Roberto de M Estevão Filho, Todd Hendry, Daniel Holstein, Jennifer Marsman, Nick Mecklenburg, Sara Malvar, Leonardo O Nunes, et al. 2024. RAG vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture. *arXiv e-prints* (2024), arXiv–2401.
[6] Fengxian Chen. 2023. Job runtime prediction of HPC cluster based on PC-Transformer. *The Journal of Supercomputing* 79, 17 (2023), 20208–20234.
[7] Xiaomeng Chen, Hui Zhang, Hanli Bai, Chunming Yang, Xujian Zhao, and Bo Li. 2020. Runtime prediction of high-performance computing jobs based on ensemble learning. In *Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications*. 56–62.
[8] Hyunjoon Cheon, Jinseung Ryu, Jaecheol Ryou, Chan Yeol Park, and Yo-Sub Han. 2023. ARED: automata-based runtime estimation for distributed systems using deep learning. *Cluster Computing* 26, 5 (2023), 2629–2641.
[9] Renato LF Cunha, Eduardo R Rodrigues, Leonardo P Tizzei, and Marco AS Netto. 2017. Job placement advisor based on turnaround predictions for HPC hybrid clouds. *Future Generation Computer Systems* 67 (2017), 35–46.
[10] Yiqin Dai, Yong Dong, Kai Lu, Ruibo Wang, Wei Zhang, Juan Chen, Mingtian Shao, and Zheng Wang. 2022. Towards scalable resource management for supercomputers. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.
[11] Yuping Fan, Paul Rich, William E Allcock, Michael E Papka, and Zhiling Lan. 2017. Trade-off between prediction accuracy and underestimation rate in job runtime estimates. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 530–540.
[12] Dror G Feitelson, Dan Tsafrir, and David Krakov. 2014. Experience with using the parallel workloads archive. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2967–2982.
[13] Eric Gaussier, David Glesser, Valentin Reis, and Denis Trystram. 2015. Improving backfilling by using machine learning to predict running times. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–10.
[14] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).
[15] Sheng-Chieh Lin, Minghan Li, and Jimmy Lin. 2023. Aggretriever: A simple approach to aggregate textual representations for robust dense passage retrieval. *Transactions of the Association for Computational Linguistics* 11 (2023), 436–452.
[16] Chun Liu, Baoqing Wang, and Yuqiang Li. 2023. Dialog generation model based on variational Bayesian knowledge retrieval method. *Neurocomputing* 561 (2023), 126878.

[17] Ju-Won Park and Eunhye Kim. 2017. Runtime prediction of parallel applications with workload-aware clustering. *The Journal of Supercomputing* 73, 11 (2017), 4635–4651.

[18] Nathalie Rauschmayr. 2015. A History-based Estimation for LHCb job requirements. In *Journal of Physics: Conference Series*, Vol. 664. IOP Publishing, 062050.

[19] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[20] Dan Tsafrir, Yoav Etsion, and Dror G Feitelson. 2007. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 18, 6 (2007), 789–803.

[21] Adam KL Wong and Andrzej M Goscinski. 2007. Evaluating the EASY-backfill job scheduling of static workloads on clusters. In *2007 IEEE International Conference on Cluster Computing*. IEEE, 64–73.

[22] Tong Ye, Lingfei Wu, Tengfei Ma, Xuhong Zhang, Yangkai Du, Peiyu Liu, Shouling Ji, and Wenhai Wang. 2023. Tram: A Token-level Retrieval-augmented Mechanism for Source Code Summarization. *arXiv preprint arXiv:2305.11074* (2023).

[23] Xunjian Yin, Baizhou Huang, and Xiaojun Wan. 2023. ALCUNA: Large language models meet new knowledge. *arXiv preprint arXiv:2310.14820* (2023).

[24] Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. 2023. Making retrieval-augmented language models robust to irrelevant context. *arXiv preprint arXiv:2310.01558* (2023).

[25] Wenhao Yu, Hongming Zhang, Xiaoman Pan, Kaixin Ma, Hongwei Wang, and Dong Yu. 2023. Chain-of-note: Enhancing robustness in retrieval-augmented language models. *arXiv preprint arXiv:2311.09210* (2023).

[26] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. 2022. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv:2207.05987* (2022).