Optimizing the Utilization of Large Language Models via Schedule Optimization: An Exploratory Study

Yueyue Liu
yueyue.liu@uon.edu.au
University of Newcastle
Newcastle, NSW, Australia

Hongyu Zhang[*]
hyzhang@cqu.edu.cn
Chongqing University
Chongqing, China

Zhiqiang Li
lizq@snnu.edu.cn
Shaanxi Normal University
Shaanxi, China

Yuantian Miao
sky.miao@newcastle.edu.au
University of Newcastle
Newcastle, NSW, Australia

Abstract

Background: Large Language Models (LLMs) have gained significant attention in machine-learning-as-a-service (MLaaS) offerings. In-context learning (ICL) is a technique that guides LLMs towards accurate query processing by providing additional information. However, longer prompts lead to higher costs of LLM service, creating a performance-cost trade-off. **Aims:** We aim to investigate the potential of combining schedule optimization with ICL to optimize LLM utilization. **Method:** We conduct an exploratory study. First, we consider the performance-cost trade-off in LLM utilization as a multi-objective optimization problem, aiming to select the most suitable prompt template for each LLM job to maximize accuracy (the percentage of correctly processed jobs) and minimize invocation cost. Next, we investigate three methods for prompt performance prediction to address the challenge of evaluating the accuracy objective in the fitness function, as the result can only be determined after submitting the job to the LLM. Finally, we apply widely used search-based techniques and evaluate their effectiveness. **Results:** The results indicate that the machine learning-based technique is an effective approach for prompt performance prediction and fitness function calculation. Schedule optimization can achieve higher accuracy or lower cost by selecting a suitable prompt template for each job, compared to simply submitting all jobs using a single prompt template, e.g., saving costs from 21.33% to 86.92% in our experiments on LLM-based log parsing. However, the performance of the evaluated search-based techniques varies across different instances and metrics, with no single technique consistently outperforming the others. **Conclusions:** This study demonstrates the potential of combining schedule optimization with ICL to improve the utilization of LLMs. However, there is still ample room for improving the searched-based techniques and prompt performance prediction techniques for more cost-effective LLM utilization.

[*]Corresponding author.

CCS Concepts

• **Software and its engineering → Search-based software engineering**; • **Applied computing → Multi-criterion optimization and decision-making**.

Keywords

Large Language Models, Schedule Optimization, Multi-objective Optimization, Search-based Techniques

ACM Reference Format:
Yueyue Liu, Hongyu Zhang, Zhiqiang Li, and Yuantian Miao. 2024. Optimizing the Utilization of Large Language Models via Schedule Optimization: An Exploratory Study. In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3674805.3686671

1 Introduction

Large Language Models (LLMs) have gained significant attention as a critical component of the rapidly expanding machine-learning-as-a-service (MLaaS) offerings in cloud computing [12, 22]. These models enable the generation and understanding of human and programming languages. An increasing number of companies offer LLM services through public APIs with varying performance and prices, such as OpenAI[1] and AI21[2]. These APIs allow users to access LLM functionalities and integrate them into various applications.

The potential of LLMs has driven the development of a variety of prompt engineering techniques, such as in-context learning (ICL) [17]. ICL has attracted considerable attention because it can guide LLMs towards accurate query processing by leveraging task instructions and demonstrative examples. By providing explicit instructions and demonstrations as input, ICL enables LLMs to make informed responses based on the given context. However, while more detailed and informative prompts increase the likelihood of correctly handling a query (regarded as a job in this paper), they also incur higher costs and lead to substantial expenses, especially when a large number of jobs are required for a task. Therefore, when adopting LLMs in practice, it is important for users to select appropriate prompts that balance performance and cost.

[1]https://openai.com
[2]https://www.ai21.com/

Schedule optimization has proven to be an effective tool in improving resource utilization in cloud computing, such as task scheduling [4, 29] and resource allocation [5, 14]. In the context of LLM usage, scheduling involves determining the most appropriate prompt template for each job to achieve the trade-off between performance and cost. Search-based techniques, a class of optimization algorithms that identify optimal or near-optimal solutions, have been successfully applied to various scheduling and resource allocation problems in cloud computing [5, 10].

However, the application of scheduling to improve the utilization of LLMs remains largely unexplored in current research. Thus, this paper presents an exploratory study that investigates the potential of combining schedule optimization with the ICL technique to improve the utilization of LLMs.

As shown in Figure 1, we first formulate the performance-cost trade-off in LLM utilization as a multi-objective optimization problem, providing allocation solutions where each job is allocated the most suitable prompt template to maximize the percentage of correctly processed jobs (referred to as accuracy in this paper) while minimizing the associated costs. Second, we construct a fitness function to evaluate the quality of generated allocation solutions, which is a crucial prerequisite for applying search-based techniques. However, calculating the accuracy objective in the fitness function poses a significant challenge, as the true accuracy can only be determined after jobs are submitted to the LLM. Therefore, we explore different methods to predict the possibility of prompt templates successfully handling jobs without actually submitting them to the LLM, including similarity-based confidence estimation (SCE), machine-learning-based prediction (MLBP), and random selection as a baseline. Third, we apply widely used search-based techniques and investigate the effectiveness of those techniques in improving LLM utilization. We select classic algorithms from various categories of search-based techniques based on their proven performance and wide applicability in optimization problems. Specifically, we investigate the following three research questions (RQs):

(1) RQ1: What kinds of prediction techniques are helpful to evaluate the accuracy objective in the fitness function without actually submitting the jobs to the LLM?
(2) RQ2: How do various search-based techniques perform in identifying optimal solutions for the prompt template allocation problem?
(3) RQ3: How generalizable are our findings across different LLMs and datasets?

To evaluate the effectiveness of schedule optimization, we select LLM-based log parsing [23, 28] as a case study. Log parsing, which extracts structured data from unstructured log files, is an essential steps of log analysis for software reliability management. For RQ1, we compare the performance of SCE, MLBP, and random selection in prompt performance prediction for fitness function calculation. We further analyze the effect of these prediction methods on the final optimization results. For RQ2, we evaluate the effectiveness of different search-based techniques in finding optimal allocation solutions for improving LLM utilization. For RQ3, we validate the generalization of our findings by extending the experiment on the text classification using the J2 model from AI21[2].

Based on the extensive experiments, we summarize the following key findings:

(1) Accurate prompt performance prediction is crucial for constructing effective fitness functions in schedule optimization.
(2) Schedule optimization, which involves selecting the most appropriate prompt template for each LLM job, can lead to significant improvements in LLM utilization compared to using an individual prompt template.
(3) The effectiveness of schedule optimization is influenced by various factors, including prediction accuracy, the efficiency of the search algorithm, the quality of the candidate prompt templates, and the capabilities of the selected LLM.

2 Background

2.1 Large Language Model

Large Language Models (LLMs) have emerged as a groundbreaking development in natural language processing (NLP). These models have the ability to understand and generate human language with remarkable proficiency, as well as to process and develop code in various programming languages. The advent of transformer-based architectures, such as BERT [15] and GPT [36], has revolutionized the way LLMs learn and process language, enabling them to demonstrate exceptional performance across a wide range of tasks [9, 37].

One of the most promising techniques to optimize LLM performance is in-context learning (ICL) [9]. ICL provides task-specific examples or demonstrations within the input prompt, allowing LLMs to adapt to new tasks without the need for fine-tuning or additional training [25, 42]. However, the effectiveness of ICL is highly dependent on the quality and relevance of the examples provided in the prompt [17]. While including more examples in the prompt can provide the LLM with additional context and information, leading to improved accuracy, it also increases the prompt length. Longer prompts require more tokens to be processed, which consequently increases the computational cost.

As research in this field progresses, developing techniques to optimize LLM utilization while considering the trade-off between performance and cost becomes increasingly important.

2.2 Scheduling Optimization

Schedule optimization is a well-established problem in optimization that involves allocating resources to tasks to maximize performance and minimize costs. It has a wide range of applications in various domains, such as production scheduling [10], project management, and cloud computing [13].

In the context of cloud computing, schedule optimization plays a crucial role in efficiently managing and allocating resources to meet the demands of various applications and users. This includes optimizing the placement of virtual machines, scheduling tasks [29] and workflows [11], and managing resource provisioning [30]. Numerous optimization techniques, such as Evolutionary Algorithms (EAs), Swarm Intelligence Algorithms (SIs), and Decomposition-based Algorithms, have been applied to solve scheduling problems in this domain [31]. For example, Chen et al.[11] present a multi-objective data placement method for workflow management in cloud infrastructure, while Luo et al.[30] investigate resource provisioning optimization using a local search algorithm.

Despite the extensive research on schedule optimization in cloud computing, its application to optimizing the utilization of LLMs

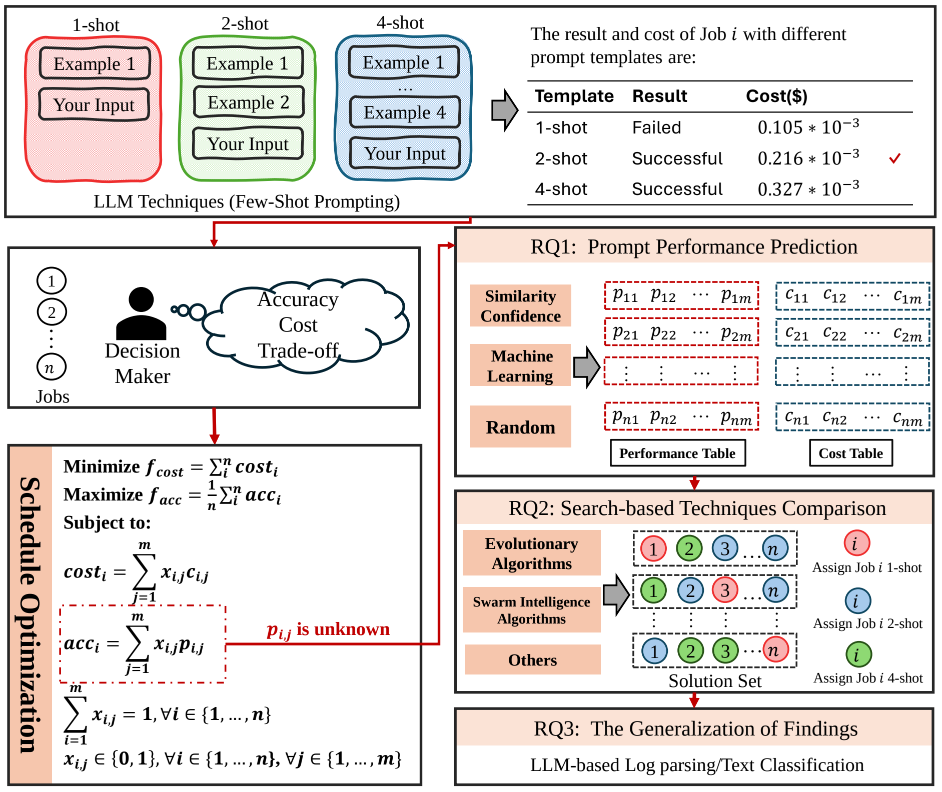
1-shot
2-shot
4-shot
Example 1
Your Input
Example 2
Example 4
LLM Techniques (Few-Shot Prompting)
The result and cost of Job i with different prompt templates are:
Template | Result | Cost($)
1-shot | Failed | $0.105 * 10^{-3}$
2-shot | Successful | $0.216 * 10^{-3}$ ✓
4-shot | Successful | $0.327 * 10^{-3}$
Jobs
Decision Maker
Accuracy
Cost
Trade-off
Schedule Optimization
Minimize $f_{cost} = \sum_i^n cost_i$
Maximize $f_{acc} = \frac{1}{n}\sum_i^n acc_i$
Subject to:
$cost_i = \sum_{j=1}^m x_{i,j} c_{i,j}$
$acc_i = \sum_{j=1}^m x_{i,j} p_{i,j}$
$p_{i,j}$ is unknown
$\sum_{i=1}^m x_{i,j} = 1, \forall i \in \{1,\dots,n\}$
$x_{i,j} \in \{0,1\}, \forall i \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\}$
RQ1: Prompt Performance Prediction
Similarity Confidence
Machine Learning
Random
Performance Table
Cost Table
RQ2: Search-based Techniques Comparison
Evolutionary Algorithms
Swarm Intelligence Algorithms
Others
Solution Set
Assign Job i 1-shot
Assign Job i 2-shot
Assign Job i 4-shot
RQ3: The Generalization of Findings
LLM-based Log parsing/Text Classification

Figure 1: A framework for improving LLM utilization through schedule optimization and the ICL technique

remains largely unexplored. This paper aims to address this challenge by exploring the integration of schedule optimization and ICL to find efficient and cost-effective prompt allocation solutions for improving LLM usage in various domains.

3 Study Methodology

The goal of this research is to investigate the possibility of improving the utilization of LLMs by integrating the ICL technique with schedule optimization. The study focuses on addressing three primary Research Questions (RQs):

RQ1: What kind of prediction techniques are helpful to evaluate the accuracy objective in the fitness function without actually submitting the jobs to the LLM?

RQ2: How do various search-based techniques perform in identifying optimal solutions for the prompt template allocation problem?

RQ3: How generalizable are our findings across different LLMs and datasets?

To tackle these RQs, the study uses LLM-based log parsing [28] as a case study. This section begins with a problem formulation, followed by an introduction to the chosen case study of LLM-based log parsing. Subsequently, specific approaches are presented to address each of the RQs. The study also defines metrics to evaluate the results and assess the effectiveness of the investigated methods.

3.1 Problem Formulation

Given a set of jobs $J = \{1, 2, \dots, n\}$ waiting to be processed on an LLM with a unit price of dp, each job i is characterized by its initial token number tn_i. A set of prompt templates $P = \{1, 2, \dots, m\}$ is available. For prompt template j, the token number is pn_j. The cost to select prompt j to process job i is:

$$cost_{i,j} = (tn_i + pn_j) \times dp \tag{1}$$

Assume $acc_{i,j}$ is the corresponding result of leveraging prompt j for job i, where $acc_{i,j} = 1$ represents that job i is processed correctly and 0 otherwise. We aim to provide a set of solutions where each solution offers a proper assignment to choose the most suitable prompts for each job, minimizing the total cost and maximizing the accuracy. Denote the decision variable as $x_{i,j}$, a binary variable set to 1 when job i is assigned to prompt j and 0 otherwise. This problem is formalized as follows:

$$\text{Minimize } f_{cost} = \sum_{i=1}^{n} cost_i \tag{2}$$

$$\text{Maximize } f_{acc} = \frac{1}{n}\sum_{i=1}^{n} acc_i \tag{3}$$

subject to:

$$cost_i = \sum_{j=1}^{m} x_{i,j} \times cost_{i,j}, \forall i \in \{1, \dots, n\} \tag{4}$$

$$acc_i = \sum_{j=1}^{m} x_{i,j} \times acc_{i,j}, \forall i \in \{1, \dots, n\} \quad (5)$$

$$\sum_{j=1}^{m} x_{i,j} = 1, \forall i \in \{1, \dots, n\} \quad (6)$$

$$x_{i,j} \in \{0, 1\}, \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \quad (7)$$

The objective functions (2) and (3) aim to optimize the cost and accuracy, respectively. Constraints (4) and (5) define the cost and accuracy for each job i. Constraint (6) ensures that each job is assigned to one prompt template. Constraint (7) represents the binary variable constraints.

3.2 A Case Study - Log Parsing

In this paper, we choose LLM-based log parsing as the case study. Log parsing typically serves as the first step toward automated log analytics for software reliability management [21, 54], aiming to parse a log message into a specific log template.

Log messages are system-generated lines of text containing event information, often including dynamic variables that vary across instances of the same event type. For example, consider the following log message: "`Block broadcast_0_piece0 stored as bytes in memory (estimated size 93.0 B, free 317.1 KB)`", LLM-based log parsing leverages the LLMs to identify and extract the dynamic variables ("`broadcast_0_piece0`", "`93.0 B`", and "`317.1 KB`") from the log message, replacing them with placeholders to create a standardized log template: "`Block {variable} stored as bytes in memory (estimated size {size}, free {memory})`".

We follow a recent LLM-based log parsing study [28], which provides six prompts (i.e., Simple, Standard, Enhance, 1-shot, 2-shot, and 4-shot) to interact with ChatGPT-3.5. The Simple, Standard, and Enhance prompts can be regarded as zero-shot learning, while 1-shot, 2-shot, and 4-shot prompts are examples of few-shot learning, which is a type of ICL technique. Few-shot learning involves providing the model with some examples to guide its output, enabling it to better understand the task at hand. Table 1 shows prompts and their costs for log parsing used for the evaluation. '[LOG]' indicates the input log message. For example, utilizing the Simple prompt incurs an additional cost of $0.00004 per log message, while employing the Enhance prompt entails an extra expense of $0.00008 per log message, above the cost for processing each log message.

We leverage log data originating from the LogPai benchmark [27, 54] to generate instances of different sizes. LogPai consists of log data from 16 different systems, including distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software.

3.3 Prediction Techniques for Prompt Performance (RQ1)

Applying search-based techniques for the prompt template allocation problem requires the definition of a fitness function. Here, the fitness function consists of two objectives: accuracy and cost. While the calculation of the cost objective (f_{cost}) is straightforward, multiplying the token number by the unit price of the chosen LLM, evaluating the accuracy objective (f_{acc}) poses a significant challenge. f_{acc} denotes the percentage of correctly processed jobs, but determining this value requires submitting the jobs to the LLM, which is impractical during the search process. To address this challenge, the first part of our study explores how to predict the possibility of each job being processed correctly by a given prompt template, which can then be used to calculate the accuracy objective without actually submitting the jobs to the LLM.

Table 1: Illustrations of prompts for log parsing

Prompts	Cost (USD)
Simple: You will be provided with a log message delimited by backticks. Please extract the log template from this log message: '[LOG]'	0.00004
Standard: You will be provided with a log message delimited by backticks. You must abstract variables with placeholders' to extract the corresponding template. Print the input log's template delimited by backticks. Log message: '[LOG]'	0.00007
Enhance: You will be provided with a log message delimited by backticks. You must identify and abstract all the dynamic variables in logs with placeholders' and output a static log template. Print the input log's template delimited by backticks. Log message: '[LOG]'	0.00008
Few-shot: You will be provided with a log message delimited by backticks. You must abstract variables with placeholders' to extract the corresponding template. For example: The template of [DEMO LOG1]' is [TEMPLATE1]'. The template of [DEMO LOG2]' is [TEMPLATE2]'. Print the input log's template delimited by backticks. Log message: '[LOG]'	depends on #examples

Note: '[LOG]' is the slot to fill in the log message to be parsed.

Instead of calculating the actual accuracy, we employ the *expected accuracy*, denoted as f'_{acc}, as a substitute in the fitness function for evaluating allocation solutions. Let $p_{i,j}$ be the predicted probability of job i being processed correctly by prompt template j. The expected accuracy of a candidate solution is calculated as follows:

$$f'_{acc} = \frac{1}{n} \sum_{i=1}^{n} p_i \quad (8)$$

$$p_i = \sum_{j=1}^{m} x_{i,j} \times p_{i,j}, \quad \forall i \in \{1, \dots, n\} \quad (9)$$

We propose the following three approaches to assess $p_{i,j}$:

(1) Similarity-based Confidence Estimation (SCE): This approach leverages historical data and similarity measures to estimate the probability of success for prompt templates. It compares a new job with similar jobs in a historical dataset and considers the outcomes of those similar jobs. Based on this comparison, the SCE method provides a confidence score for each candidate template, indicating its likelihood of successfully handling the new job. In our implementation, we use Maximum Mean Discrepancy (MMD) as the similarity measure to calculate the distance between jobs.
(2) Machine-learning-based Prediction (MLBP): This approach utilizes historical data and learning algorithms to predict the performance of prompt templates. By training a predictive model on a dataset of jobs, prompt templates, and their corresponding success/failure outcomes, we can estimate the likelihood of a template successfully processing a new job without actually submitting it to the LLM. We employ XGBoost, a gradient-boosting algorithm, as the learning algorithm for this approach.

Table 2: Overview of the evaluated search-based techniques

Algorithm	Category	Description	Application
NSGA-II	EA	An extension of the Genetic Algorithm (GA) for multi-objective optimization. Uses non-dominated sorting and crowding distance to maintain diversity.	Resource provisioning [47], resource allocation [45], workflow scheduling [26], data placement [46], and energy-efficient scheduling [40] in cloud computing.
RNSGA-II	EA	An extension of NSGA-II designed for many-objective optimization problems. Uses reference points to guide the search process.	Resource allocation [33, 35, 48] and virtual machine placement [34] in cloud computing.
SPEA2	EA	An evolutionary algorithm that maintains an external archive of non-dominated solutions and uses a fine-grained fitness assignment strategy based on dominance strength and density information.	Resource provisioning [50], workflow scheduling, energy-efficient scheduling in cloud computing [52].
MOPSO	SI	An extension of Particle Swarm Optimization (PSO) for multi-objective problems. Uses an external archive to store non-dominated solutions and guide the search process.	Resource allocation [16], task scheduling [3], workflow scheduling [20, 39], and load balancing [32] in cloud computing.
MOACO	SI	An extension of Ant Colony Optimization (ACO) for multi-objective problems. Uses multiple pheromone matrices to optimize different objectives simultaneously.	Virtual machine placement [18], workflow scheduling [49], service composition [53], and load balancing [38] in cloud computing.
MOEA/D	Decomposition	A decomposition-based evolutionary algorithm that decomposes a multi-objective problem into a set of scalar subproblems and optimizes them simultaneously.	Resource allocation [24], task scheduling [41], and virtual machine placement [19] in cloud computing.
MOEA/D-GEN	Decomposition	A variant of MOEA/D that incorporates generalized subproblem-dependent heuristics for offspring generation.	Scalable multi-objective test problems in software engineering [44], multi-objective optimizatin [43].

(3) Random estimation: This approach serves as a baseline for comparison, randomly generating the possibility of each job being processed by a given template without considering any specific criteria or historical data.

3.4 Search-based Techniques (RQ2)

There are various search-based techniques for multi-objective optimization, such as Evolutionary Algorithms (EAs), Swarm Intelligence Algorithms (SIs), Decomposition-based Algorithms, etc. To provide a comprehensive analysis of these techniques in the context of prompt template allocation optimization, we select representative algorithms as shown in Table 2.

We began by choosing fundamental algorithms such as NSGA-II (an EA algorithm), MOPSO (an SI algorithm), and MOEA/D (a Decomposition-based algorithm). To enhance the scope of our study, we expanded our selection to include additional algorithms and variants, such as SPEA2 from EAs, MOACO from SIs, and RNSGA-II and MOEA/D-GEN as variations of NSGA-II and MOEA/D, respectively.

To apply these algorithms to the prompt template allocation problem, we first define the solution representation and the objective functions. Each solution is represented as a vector of integers, where each element corresponds to a job and its value indicates the index of the assigned prompt template. We build the fitness function consisting of the two objectives, which are described in Section 3.1, to evaluate the invocation cost and accuracy of a given allocation solution. The optimization begins by initializing a population of random solutions. In each iteration, the algorithms generate new solutions by applying various operators, such as mutation, crossover, or swarm-based movements, depending on the specific algorithm. The generated solutions are then evaluated using the fitness function, and the best solutions are selected to form the next generation or to update the positions of the swarm. All algorithms repeat the generation and evaluation process until a stopping criterion is met. In this study, we define the same iteration number for all evaluated search-based techniques.

3.5 Evaluation Metrics

This paper evaluates the quality of the obtained solutions from two aspects. First, we directly compare the cost and accuracy of the allocation solutions. Second, to assess the quality of the generated solutions, we introduce three widely used metrics in multi-objective optimization problems: IGD, Δ, and M_n [6].

Objectives (cost and accuracy). We compare the optimization objectives for a straightforward assessment of the solutions in terms of cost and accuracy. All algorithms generate a set of solutions rather than a single solution. For comparison purposes, we select the solution that achieves the highest accuracy and report it. We also report the cost reduction achieved by the best scheduling solution compared with submitting all jobs using a single prompt template. Recall the two objectives:

- f_{cost}: the cost of invoking LLM APIs at the unit of USD (see Equation 2)
- f_{acc}: the percentage of jobs that have been processed correctly (see Equation 3)

In multi-objective optimization, a solution is considered **non-dominated** if no other solution is better in all objectives. The set of non-dominated solutions forms the **Pareto front**, representing the best trade-offs between the competing objectives. The following metrics are used to evaluate the quality of the obtained solution set in relation to the Pareto front.

Inverted Generational Distance (IGD). IGD measures how close the obtained solutions are to the optimal solutions, evaluating the goodness of the obtained solution set [6]. Let the solution set $\Lambda = \{y_1, y_2, \ldots, y_{|\Lambda|}\}$ be the Pareto front for the problem and $\hat{\Lambda} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{|\hat{\Lambda}|}\}$ be the solutions obtained by an algorithm. We can define IGD by:

$$IGD(\hat{\Lambda}, \Lambda) = \frac{1}{|\Lambda|}\sqrt{\sum_{y \in \Lambda}\left(\min_{\hat{y} \in \hat{\Lambda}} d(\hat{y}, y)\right)^2}$$

Table 3: Comparisons of all algorithms with different fitness function in terms of accuracy, IGD, Δ, and M_n

Algorithms	Fitness Function	Highest Accuracy				IGD				Δ				M_n			
		Max	Min	Median	Avg	Max	Min	Median	Avg	Max	Min	Median	Avg	Max	Min	Median	Avg
NSGA-II	SCE	0.9405	0.0609	0.6257	0.5792	0.2767	0.0603	0.1200	0.1259	0.9993	0.9626	0.9915	0.9888	15	2	5	5
	MLBP	**0.9541**	**0.3255**	0.6268	0.6529	0.1994	0.0615	0.0987	0.1143	0.9918	0.8635	0.9722	0.9589	72	3	15	21
	random	0.9210	0.0937	0.6297	0.5893	0.2557	0.0685	0.1054	0.1242	0.9828	0.9647	0.9782	0.9749	12	3	6	6
SPEA2	SCE	0.9474	0.1682	0.6715	0.6480	0.2163	0.0802	0.1093	0.1268	0.9198	0.7757	0.8280	0.8424	21	5	12	11
	MLBP	0.9502	0.2705	**0.6762**	0.6601	0.2103	0.0798	0.1088	0.1250	0.9134	0.7672	**0.8107**	**0.8176**	29	5	13	13
	random	0.9453	0.2432	0.6733	0.6477	0.2106	0.0799	0.1087	0.1246	**0.8980**	0.7978	0.8515	0.8485	20	4	9	10
MOPSO	SCE	0.8957	0.0638	0.6283	0.5726	0.2726	0.0518	0.0988	0.1151	1.0008	0.8571	0.9696	0.9541	29	1	5	9
	MLBP	0.9090	0.1344	0.6253	0.5851	0.2194	0.0548	0.0924	0.1073	1.0006	0.7837	0.8898	0.9065	30	2	10	13
	random	0.9015	0.0790	0.6246	0.5674	0.2595	0.0563	0.0978	0.1166	0.9882	0.9384	0.9691	0.9686	19	2	9	9
MOACO	SCE	0.9400	0.2065	0.6610	0.6440	0.2329	0.0920	0.1216	0.1383	0.9290	0.8440	0.8803	0.8821	10	5	8	7
	MLBP	0.9480	0.2667	0.6678	**0.6602**	0.2335	0.0921	0.1230	0.1381	0.9188	0.8351	0.8774	0.8778	23	**6**	9	11
	random	0.9410	0.2143	0.6616	0.6415	0.2368	0.0939	0.1227	0.1389	0.9267	0.8509	0.8890	0.8855	12	5	7	7
MOEA/D	SCE	0.9098	0.0217	0.6169	0.5628	0.3083	0.0504	0.1000	0.1219	1.0563	0.8492	0.9907	0.9701	52	2	7	11
	MLBP	0.9484	0.2727	0.6657	0.6167	**0.1684**	0.0489	**0.0872**	**0.0988**	1.0215	0.7634	0.8963	0.9046	**82**	4	23	**27**
	random	0.8960	0.0513	0.6254	0.5572	0.2812	0.0533	0.1006	0.1175	0.9862	0.9437	0.9647	0.9646	22	4	11	12
RNSGA-II	SCE	0.8903	0.0264	0.6121	0.5532	0.3039	0.0563	0.1278	0.12695	1.0000	0.9828	0.9957	0.9951	18	2	3	5
	MLBP	0.9354	0.2644	0.6075	0.6063	0.2486	**0.0392**	0.0964	0.1051	0.9955	0.9251	0.9619	0.9632	51	4	19	19
	random	0.9128	0.0651	0.6270	0.5783	0.2725	0.0544	0.1019	0.1168	0.9720	0.9291	0.9489	0.9487	17	4	9	10
MOEA/D-GEN	SCE	0.9113	0.0213	0.6150	0.5623	0.3086	0.0477	0.1033	0.1229	1.0474	0.8301	0.9882	0.9632	54	3	8	13
	MLBP	0.9510	0.2788	0.6618	0.6158	0.1701	0.0459	0.0880	0.0989	1.0760	**0.7409**	0.9039	0.9088	80	2	**25**	**27**
	random	0.8967	0.0499	0.6238	0.5567	0.2823	0.0524	0.0994	0.1173	0.9831	0.9315	0.9632	0.9613	24	3	13	13

where $|\hat{\Lambda}|$ denotes the number of solutions in the obtained solution set. $d(\hat{y}, y)$ is the Euclidean distance between a solution $\hat{y}$ in $\hat{\Lambda}$ and the nearest solution y in the true Pareto front Λ. A lower value of IGD represents a better performance, indicating the obtained solution set has a better distribution and better approximation to the reference set. The most promising IGD value is 0, which means that the set of obtained solutions is equal to the reference point set.

Δ **metric**. Δ assesses the diversity and uniformity of the solutions distribution along the Pareto front by measuring Euclidean distances between consecutive solutions and comparing them to the average distance [6].

$$\Delta(\hat{\Lambda}) = \frac{d_f + d_l + \sum_{z=1}^{|\hat{\Lambda}|-1} |d_z - \bar{d}|}{d_f + d_l + (|\hat{\Lambda}| - 1)\bar{d}}$$

where d_f and d_l are the Euclidean distances between the extreme solutions and their nearest neighbours in the obtained solution set $\hat{\Lambda}$. d_z denotes the Euclidean distance between the zth solution $\hat{y}_z$ and its next solution $\hat{y}_{z+1}$ in the obtained solution set $\hat{\Lambda}$, sorted in ascending order of a chosen objective. $\bar{d}$ represents the mean value of the distances d_z. A smaller Δ metric indicates a higher diversity and distribution of solutions.

The number of non-dominated solutions M_n: M_n is selected as the cardinality indicator, quantifying the number of non-dominated solutions generated by an algorithm. A larger value of M_n indicates that there are more non-dominated solutions in the obtained solutions set $\hat{\Lambda}$. M_n is calculated by:

$$M_n = \{\hat{\Lambda} - \{\hat{y} \in \hat{\Lambda} | y \in \Lambda : \hat{y} \prec y\}\}$$

where $\hat{y} \prec y$ means that solution y dominates solution $\hat{y}$.

4 Results and Analysis

4.1 RQ1: Prompt Performance Prediction

To evaluate the performance of the tested methods, we use LLMs-based log parsing as the study case. Log data from each system is treated as an individual instance, leading to 16 instances in total. All the evaluated search-based techniques are integrated with the three prediction components ($7 \times 3 = 21$ algorithms). We run experiments on instances, testing all 21 algorithms and repeating 10 times to mitigate the impact of randomness, resulting in a total of 3,360 experimental runs (21 algorithms × 16 instances × 10 repetitions).

Table 4: Prediction accuracy under different training data sizes on Mac instance

training size	Simple	Standard	Enhance	1-shot	2-shot	4-shot
1%	0.6779	0.7279	0.7293	0.7114	0.6064	0.6757
5%	0.9007	0.8414	0.8771	0.8800	0.8207	0.8429
10%	0.9329	0.8843	0.8943	0.8964	0.8643	0.8888
30%	0.9450	0.9264	0.9307	0.9321	0.9229	0.9286

In Table 3, we report the highest accuracy obtained, IGD, Δ, and the number of non-dominated solutions (M_d) by all algorithms combined with three prediction methods. Due to the limited space, the mean value of 16 instances is reported. The completed results are provided on our webpage[3]. The results shown in Table 3 reveal that the MLBP method is the most effective approach for prompt performance prediction, which is used to evaluate the accuracy objective in the fitness function. By providing more accurate predictions of prompt performance, the MLBP method enables the fitness function to better assess the quality of candidate solutions, resulting in the highest accuracy across the majority of algorithms. In contrast, the random method consistently yields the lowest accuracy, while the SCE method falls between MLBP and random. The MLBP method also demonstrates superiority in terms of the IGD, Δ. It achieves the lowest or among the lowest IGD values, indicating better convergence and diversity, and the lowest or near-lowest Δ values, demonstrating its effectiveness in providing a diverse solution set. Furthermore, the MLBP method consistently achieves the best M_n values, representing the highest number of non-dominated

[3]https://github.com/LLMs-EffiUse-Lab/Sched-ICL-Empirical

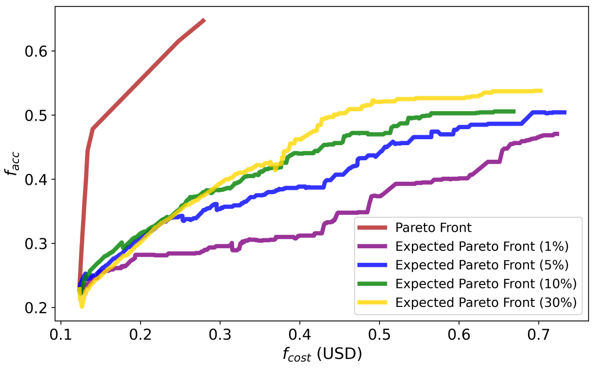

Figure 2: Expected Pareto front for varying training data sizes on Mac instance

solutions obtained across different algorithms. We summarize our findings for RQ1 as follows:

(1) In conclusion, MLBP outperforms the other tested methods in prompt performance prediction. It consistently enables algorithms to achieve higher accuracy, better IGD, Δ, and M_n, indicating superior convergence and diversity in provided solutions. The SCE method generally falls between MLBP and random in terms of performance, while the random method consistently underperforms across all metrics. To further explore the factors in fitness function that affect optimization and identify potential improvements, we focus on the MLBP method with subsequent analyses.

(2) Prompt performance prediction plays a crucial role in guiding the search algorithms, as more accurate prediction leads to an expected Pareto Front that better approximates the True Pareto Front. Figure 2 illustrates the impact of varying training data sizes on generating the Pareto Front. The figure plots the true Pareto Front (optimal solutions in the real world) in red line, and the Expected Pareto Fronts (optimal solutions obtained by using predicted accuracy) for different sampling percentages (1%, 5%, 10%, and 30%) of the training data, shown in purple, blue, green, and yellow lines, respectively. As the training data sampling percentage increases from 1% to 30%, the Expected Pareto Front converge towards the true Pareto Front. This observation, supported by the prediction accuracy results in Table 4, suggests that utilizing larger training data sizes enhances the prediction accuracy for each label, consequently improving the estimation of the Pareto Front. The improved prediction accuracy enables the algorithms to more effectively identify near-optimal solutions that better approximate the true optimal trade-offs.

(3) The importance of prediction accuracy on different prompt templates varies. Upon closer examination, it becomes apparent that the significance of prediction accuracy varies across different prompt templates. While we report average results due to space constraints, it is important to note that the impact of prediction accuracy is more pronounced for cheaper prompt templates. Accurate predictions on these less expensive prompts contribute to significant cost savings by preventing the search algorithm from allocating jobs to more expensive alternatives. Moreover, correct predictions help avoid scenarios where jobs are assigned to cheaper machines that cannot handle them effectively, thereby improving overall performance.

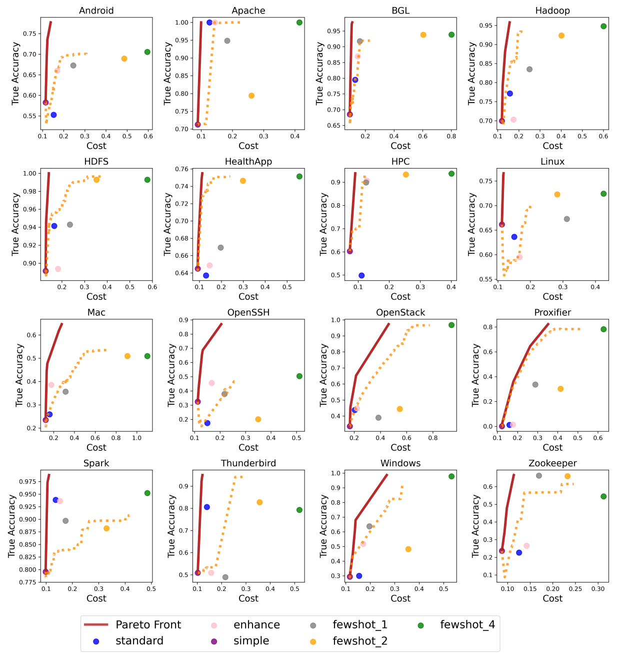

Figure 3: The comparison of individual solution from the optimal solution by schedule optimization

(4) The prediction techniques have a huge impact on guiding search algorithms, and there is still ample room for improvement in prediction accuracy. As shown in Figure 3, the red points represent the best solutions from schedule optimization with perfect knowledge of job performance on each prompt template. Orange points indicate optimal solutions using predicted accuracy values, while blue points show results from using a single prompt template without schedule optimization. The expected Pareto front, which is the set of best solutions obtained using predicted accuracy, deviates significantly from the actual Pareto front in some instances, such as OpenSSH and Spark. This indicates that prediction accuracy strongly influences the effectiveness of search algorithms in finding optimal solutions, suggesting that there is still considerable room for improving prediction accuracy to enhance the performance of search algorithms further.

Answer to RQ1: In summary, the MLBP method is the most effective approach for prompt performance prediction for calculating the accuracy objective without actually submitting the jobs to the LLM. However, there is still room for improvement. The effectiveness of this method relies on the accuracy of the prediction model. Some prompt templates may be more sensitive to prediction accuracy than others.

4.2 RQ2: Performance of Schedule Optimization

From the results, we have obtained the following conclusions.

Table 5: Cost savings by schedule optimization compared with the individual prompt (f_{cost} and f_{acc} are the cost and accuracy of the solution)

Metrics	Best Individual Prompt Template			Optimal Schedule in Real World		Optimal Schedule by Current Prediction	
Dataset	Prompt	f_{acc}	f_{cost}	f_{cost}	Saving	f_{cost}	Saving
Android	4-shot	0.6961	0.5964	0.1215	79.63%	0.2328	60.97%
Apache	standard	1	0.1266	0.0996	21.33%	0.1429	\
BGL	4-shot	0.9352	0.8009	0.1022	86.92%	0.1932	75.27%
Hadoop	4-shot	0.9469	0.6006	0.1519	74.71%	0.2906	51.62%
HDFS	4-shot	0.9929	0.5781	0.1424	69.43%	0.2954	36.58%
HealthApp	4-shot	0.7499	0.5565	0.1120	79.87%	0.1669	70.01%
HPC	4-shot	0.9383	0.4009	0.0908	77.35%	0.1253	68.75%
Linux	4-shot	0.7171	0.4256	0.1152	70.98%	\	\
Mac	2-shot	0.5080	0.9115	0.1681	82.65%	0.5213	46.20%
OpenSSH	4-shot	0.5114	0.5119	0.1210	76.36%	0.2700	47.26%
OpenStack	4-shot	0.9657	0.9414	0.4545	51.72%	0.6541	30.52%
Proxifier	4-shot	0.7768	0.6285	0.3288	47.68%	0.3749	40.35%
Spark	4-shot	0.9536	0.4845	0.1040	78.53%	0.2284	52.86%
Thunderbird	2-shot	0.8286	0.3568	0.1151	67.74%	0.2506	29.76%
Windows	4-shot	0.9794	0.5324	0.2652	50.19%	\	\
Zookeeper	1-shot	0.6649	0.1696	0.1139	32.84%	\	\

(1) Schedule optimization can maintain accuracy while reducing costs or improve accuracy by selecting the best template for each job. Table 5 shows the cost savings achieved by schedule optimization compared to using individual prompt templates. Schedule optimization achieves the same accuracy at lower costs or higher accuracy by selecting the most suitable template for each job. This is because (a) cheaper templates may handle certain jobs as effectively as more expensive ones, and (b) some jobs require specific templates for accurate processing, enhancing overall accuracy. However, improvement is still limited by the accuracy of prompt performance predictions.

(2) The performance of the evaluated searched-based techniques varies across different instances and metrics, with no single algorithm consistently outperforming the others. Table 6 presents the highest accuracy, IGD, Δ, and the number of non-dominated solutions M_n obtained by each algorithm across all instances. Based on the results, no single algorithm emerges as the clear winner across all instances and metrics. For example, RNSGA-II achieves the best IGD values and provides the solutions set that are the most closed to the optimal solutions in most instances, while SPEA2 achieves the best diversity and obtains the smallest Δ values in the majority of cases. MOEA/D-GEN performs relatively well providing the highest M_n values in 5 out of 16 instances. This highlights the importance of considering multiple metrics when evaluating the performance of algorithms. Also, the effectiveness of algorithms may vary depending on the specific problem.

(3) There is still ample improvement room for the searched-based technique in improving LLM utilization. The improvement extent through schedule optimization is not only limited by the prediction techniques but also by the capabilities of the searched-based techniques. As shown in Figure 4, the approximate Pareto front by all algorithms is still far away from the Pareto front and Expected Pareto front. Our results show that all evaluated algorithms have not yet achieved the best optimal solutions. Some algorithms, such as NSGA-II and MOPSO, get trapped in suboptimal solutions and fail to explore more diverse solutions, especially when the predicted accuracies for different jobs are similar. In such cases, the algorithms may struggle to identify the globally optimal solutions without additional problem-specific information to guide the search process. This highlights the need for further research and development of more advanced intelligent algorithms that can effectively navigate the search space and find optimal solutions to complex optimization problems.

(4) The performance of the candidate prompt templates and the selected LLM significantly limits the potential improvements achievable through schedule optimization. Although schedule optimization can achieve cost savings or accuracy improvement, its effectiveness is ultimately determined by the performance of the underlying prompt templates and the LLM. This limitation is exemplified by the Zookeeper instance, where the highest accuracy achieved through schedule optimization is 0.6649. When considering the Zookeeper instance, the prompt templates (Simple, Standard, Enhance, 1-shot, 2-shot, and 4-shot) can correctly process 0.2334, 0.2325, 0.2721, 0.6649, 0.6614, and 0.5429 of the jobs, respectively. In this case, 1-shot prompt template achieves the highest accuracy, and there are no jobs that 1-shot prompt template cannot handle but other templates can. Consequently, the accuracy cannot be improved further. However, cost savings can still be realized through schedule optimization, as some jobs can be processed by cheaper prompt templates, such as the Simple template, compared to the more expensive 1-shot prompt template.

Answer to RQ2: Schedule optimization can provide a suitable prompt allocation, achieving higher accuracy or lower cost compared to simply submitting all jobs using a certain prompt template. However, the performance of the currently evaluated search-based techniques still falls short of the optimal schedule solutions, indicating room for further improvement. The potential improvements achievable through schedule optimization are limited by the performance of the candidate prompt templates and the selected LLM.

4.3 RQ3: The Generalization of Findings

To ensure that the schedule optimization is not limited to a particular task or LLM, we extend our study to a text classification task using a different LLM, namely the J2-Mid. The J2 (Jurassic-2) model is a family of state-of-the-art large language models developed by AI21 Labs [2]. It offers significant improvements in quality, new capabilities such as zero-shot instruction-following, reduced latency, and multi-language support. Text classification is a fundamental problem in NLP that aims to assign predefined categories to text documents [2]. We choose the overruling dataset [51], which consists of legal documents. The task is to classify whether a sentence in a legal document contains an overruling decision or not.

The results of the experiment are presented in Table 7. Similar to the findings on the LLM-based log parsing, the best values in terms of Accuracy, IGD, Δ, and M_d are achieved by the MLBP method. This is consistent with the findings from RQ1, where MLBP was found to be the most effective approach for developing a fitness function that can evaluate candidate solutions without actually submitting them to the LLM.

Furthermore, the performance of the evaluated algorithms varies across different metrics, with no single algorithm consistently outperforming the others. This aligns with the findings from RQ2,

Table 6: Comparisons of all algorithms in terms of accuracy, IGD, Δ, and M_n

Metrics	Highest Accuracy							IGD						
Dataset	NSGA-II	SPEA2	MOPSO	MOACO	MOEA/D	RNSGA-II	MOEA/D -GEN	NSGA-II	SPEA2	MOPSO	MOACO	MOEA/D	RNSGA-II	MOEA/D -GEN
Android	0.6237	0.6486	0.5975	**0.6574**	0.5887	0.6044	0.5852	0.0924	0.1211	**0.0820**	0.1331	0.0861	0.0887	0.0873
Apache	0.9212	**0.9252**	0.8748	0.9029	0.9175	0.8657	0.9175	0.0989	0.0983	0.0618	0.1044	0.0581	0.0750	**0.0542**
BGL	0.8344	0.8698	0.7680	**0.8710**	0.7579	0.8084	0.7552	0.0975	0.1366	0.0896	0.1601	0.0882	**0.0828**	0.0887
Hadoop	**0.8451**	0.8306	0.7701	0.8287	0.8326	0.8140	0.8262	0.0958	0.1103	0.0952	0.1234	0.0830	**0.0751**	0.0839
HDFS	**0.9541**	0.9502	0.9090	0.9480	0.9484	0.9354	0.9510	0.0615	0.0812	0.0557	0.0976	0.0527	**0.0453**	0.0524
HealthApp	**0.7322**	0.6902	0.6615	0.6999	0.6954	0.7104	0.6990	0.0716	0.0798	0.0548	0.0925	0.0489	**0.0392**	0.0459
HPC	0.7393	0.8165	0.7085	**0.8346**	0.6604	0.7121	0.6529	0.0946	**0.0898**	0.0982	0.1046	0.1204	0.0978	0.1252
Linux	0.6298	0.6730	0.6531	**0.6782**	0.6710	0.6105	0.6706	0.0984	0.0985	0.0719	0.1102	0.0706	0.1055	**0.0677**
Mac	0.3740	0.3870	0.2684	**0.3906**	0.2727	0.3346	0.2881	0.1737	0.2103	0.1639	0.2335	0.1681	**0.1444**	0.1679
OpenSSH	0.3255	0.3631	0.3633	**0.3704**	0.3440	0.2644	0.3418	0.1994	0.2037	**0.1549**	0.1843	0.1684	0.2486	0.1701
OpenStack	**0.5854**	0.5264	0.4164	0.5268	0.5143	0.4713	0.5283	0.1426	0.1540	0.1217	0.1839	0.1027	**0.0949**	0.1017
Proxifier	**0.3962**	0.2705	0.1344	0.2667	0.2848	0.2920	0.2788	0.1434	0.1765	0.2194	0.1899	0.1517	**0.1360**	0.1519
Spark	0.8779	**0.9150**	0.8500	0.9067	0.8267	0.8575	0.8251	0.0642	0.0835	0.0588	0.0921	0.0660	**0.0584**	0.0670
Thunderbird	0.5874	0.6793	0.5229	0.6387	**0.6862**	0.5506	0.6719	0.1725	0.1438	0.2114	0.1697	**0.1330**	0.1891	0.1359
Windows	**0.5663**	0.5571	0.4674	0.5538	0.4178	0.4602	0.4110	0.1138	0.1073	**0.0718**	0.1225	0.1012	0.0979	0.1027
Zookeeper	0.4534	0.4594	0.3956	**0.4894**	0.4490	0.4086	0.4497	0.1088	0.1056	0.1056	0.1072	0.0821	0.1033	**0.0791**
Metrics	Δ							M_n						
Dataset	NSGA-II	SPEA2	MOPSO	MOACO	MOEA/D	RNSGA-II	MOEA/D -GEN	NSGA-II	SPEA2	MOPSO	MOACO	MOEA/D	RNSGA-II	MOEA/D -GEN
Android	0.9823	**0.7833**	0.9974	0.8667	0.9822	0.9600	0.9854	10	9	3	6	5	**18**	4
Apache	0.9629	0.9134	**0.8721**	0.8926	0.8926	0.9807	0.9002	16	5	22	7	**23**	13	20
BGL	0.9784	**0.7779**	0.9910	0.8580	0.9820	0.9794	0.9675	7	10	6	7	19	9	**23**
Hadoop	0.9616	0.7921	0.8631	0.8714	**0.7672**	0.9275	0.7742	14	12	19	10	**43**	21	**43**
HDFS	0.9873	**0.7962**	0.8299	0.8591	0.8211	0.9620	0.8144	5	9	12	9	23	16	**27**
HealthApp	0.9592	0.7836	0.8114	0.8585	0.7634	0.9417	**0.7409**	5	13	11	16	43	26	**48**
HPC	0.9918	**0.8321**	1.0006	0.8968	0.9999	0.9946	0.9997	7	**16**	2	8	7	10	12
Linux	0.9856	0.8377	**0.8204**	0.8803	0.8390	0.9763	0.8443	3	7	5	7	9	4	**10**
Mac	0.9230	**0.7672**	0.9795	0.8351	1.0215	0.9265	1.0760	**25**	14	9	10	7	22	11
OpenSSH	0.9707	**0.8460**	1.0002	0.9058	0.9936	0.9955	0.9988	**19**	13	2	6	4	9	2
OpenStack	0.8635	**0.7692**	0.7837	0.8393	0.9000	0.9251	0.9076	**50**	20	19	17	44	25	41
Proxifier	0.9086	0.8292	0.8734	0.8987	**0.7733**	0.9618	0.7906	72	29	24	23	**82**	51	80
Spark	0.9837	**0.7960**	0.9970	0.8745	0.9950	0.9925	0.9899	6	**7**	3	**7**	6	6	6
Thunderbird	0.9794	**0.8489**	0.9166	0.9014	0.8868	0.9573	0.8955	22	8	7	9	**40**	22	34
Windows	0.9306	**0.8251**	0.8609	0.8872	0.9773	0.9839	0.9761	**41**	22	28	13	23	19	26
Zookeeper	0.9737	0.8830	0.9061	0.9188	**0.8791**	0.9468	**0.8791**	32	16	30	13	**48**	37	46

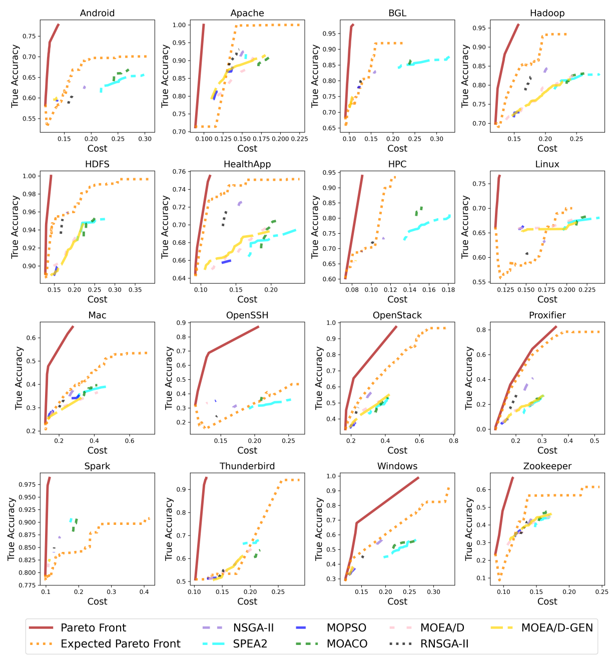

Figure 4: The approximate Pareto front by all algorithms

emphasizing the importance of considering multiple metrics when evaluating optimization algorithms.

Additionally, the Pareto fronts consistently dominate individual prompt template solutions, meaning schedule optimization achieves either higher accuracy for the same cost or lower cost for the same accuracy. These results demonstrate the effectiveness of schedule optimization in enhancing accuracy and reducing costs, suggesting its benefits extend beyond a specific task or LLM.

Answer to RQ3: The findings from the text classification task using the J2-Mid model are consistent with those from the log parsing task using ChatGPT-3.5, demonstrating the robustness and generalizability of our findings.

5 Future Research Directions

Based on the exploratory study, we point out possible future research directions, which could further leverage the potential of schedule optimization for achieving more cost-effective utilization of LLMs in real-world scenarios.

First, developing more effective and efficient optimization algorithms tailored to the schedule optimization combined with ICL techniques to improve LLMs utilization is a promising direction. While we use classic multi-objective optimization algorithms, there is potential to design algorithms that leverage unique characteristics and constraints. This could involve incorporating domain-specific knowledge, exploring alternative optimization strategies, or developing hybrid approaches that combine multiple algorithms.

Table 7: Comparisons of all algorithms in terms of accuracy, IGD, Δ, and M_n on text classification task

Algorithms	Fitness	Highest Accuracy				IGD				Δ				M_n			
	Function	Max	Min	Median	Avg	Max	Min	Median	Avg	Max	Min	Median	Avg	Max	Min	Median	Avg
NSGA-II	SCE	0.6356	0.6131	0.6991	0.6232	0.2261	0.2227	0.2321	0.2247	0.9997	0.9971	0.9979	0.9988	5	3	2	4
	MLBP	0.7222	0.6766	0.6997	0.6991	0.2327	0.2254	0.2296	0.2299	1.0000	0.9914	0.9945	0.9943	7	2	**6**	5
	random	0.7024	0.6753	0.6839	0.6875	0.2344	0.2299	0.2314	0.2314	0.9968	0.9972	0.9992	0.9991	6	2	3	3
SPEA2	SCE	0.6700	0.6561	0.6660	0.6648	0.2113	0.2039	0.2044	0.2065	0.9808	0.9729	0.9762	0.9758	9	2	4	5
	MLBP	0.6799	0.6581	0.6687	0.6682	0.2042	0.1961	0.2013	0.2013	**0.9655**	0.9520	**0.9561**	**0.9578**	8	3	**6**	5
	random	0.6739	0.6541	0.6673	0.6664	0.2100	0.1970	0.2068	0.2057	0.9816	0.9732	0.9769	0.9772	6	2	4	4
MOPSO	SCE	0.7302	0.6138	0.7692	0.6373	0.2229	0.2002	0.2033	0.2185	1.0003	0.9612	0.9522	0.9954	**14**	1	1	5
	MLBP	0.7923	0.7302	0.7579	0.7596	0.2234	0.2034	0.2210	0.2190	1.0003	**0.9420**	0.9964	0.9885	**14**	2	3	5
	random	0.6905	0.6270	0.6468	0.6468	0.2521	0.1969	0.2435	0.2393	1.0024	0.9694	1.0001	0.9959	4	1	3	3
MOACO	SCE	0.6614	0.6415	0.6548	0.6512	0.2072	0.1960	0.2003	**0.2006**	0.9797	0.9652	0.9643	0.9718	8	3	3	5
	MLBP	0.6812	0.6627	0.6726	0.6714	**0.2038**	0.1964	0.2001	**0.2006**	0.9784	0.9624	0.9711	0.9717	9	**4**	**6**	**7**
	random	0.6634	0.6429	0.6528	0.6542	0.2055	0.1991	0.2001	0.2013	0.9794	0.9712	0.9752	0.9746	7	2	3	3
MOEA/D	SCE	0.6144	0.5913	0.7798	0.6058	0.2237	0.2154	0.2174	0.2217	1.0214	1.0000	0.9941	1.0052	5	1	1	3
	MLBP	**0.8056**	0.7427	0.7659	0.7720	0.2227	0.2140	**0.1842**	0.2183	1.0118	0.9882	0.9988	0.9997	10	2	5	5
	random	0.6323	0.6091	0.6210	0.6207	0.2375	0.2071	0.2154	0.2198	1.0346	1.0001	1.0089	1.0109	2	1	1	1
RNSGA-II	SCE	0.6382	0.6038	0.7460	0.6216	0.2266	0.2224	0.2487	0.2250	1.0004	0.9982	0.9998	0.9992	9	1	1	4
	MLBP	0.7302	0.7037	0.7143	0.7137	0.2373	0.2282	0.2299	0.2310	0.9948	0.9866	0.9895	0.9897	7	1	5	4
	random	0.7249	0.6938	0.7116	0.7133	0.2415	0.2332	0.2377	0.2374	1.0002	0.9991	0.9994	0.9995	3	1	1	1
MOEA/D-GEN	SCE	0.6204	0.5853	0.7725	0.6068	0.2249	0.2153	0.2155	0.2204	1.0218	0.9999	0.9961	1.0064	6	1	1	3
	MLBP	0.7950	**0.7586**	**0.7831**	**0.7799**	0.2208	**0.1845**	0.1847	0.2164	1.0117	0.9870	0.9989	0.9988	10	3	**6**	**7**
	random	0.6528	0.6184	0.6276	0.6292	0.2455	0.2146	0.2051	0.2125	1.0232	0.9983	1.0075	1.0088	2	1	1	1

Second, investigating the scalability and real-time applicability of schedule optimization is crucial for practical deployment. Future research should focus on developing efficient algorithms and frameworks that can handle large-scale datasets and real-time scheduling scenarios. This may involve exploring distributed computing techniques, parallel processing, or incremental optimization approaches to apply schedule optimization in real-world settings.

Third, integrating schedule optimization with other techniques for improving LLM utilization, such as prompt engineering and transfer learning, could lead to further advancement. Exploring the synergies between these techniques and schedule optimization may unlock new possibilities for enhancing the efficiency and effectiveness of LLMs in various applications.

Finally, exploring the uncertain generation cost of LLMs is also a crucial direction for future work. Incorporating the generation cost into the optimization process would provide a more comprehensive assessment of the overall cost-effectiveness of schedule optimization. This could involve developing cost models that consider both the input token cost and the generation cost, as well as investigating techniques to handle the uncertainty associated with the generation cost.

6 Threats to Validity

Internal Validity. This study assumes costs are primarily associated with input tokens and does not account for generation costs, which depend on the length of the output and are uncertain. This may threaten the internal validity of our findings, as actual LLM costs can vary. We plan to explore generation costs in future work. Another potential threat is using different packages to implement the compared algorithms. We utilize NSGA-II, RNSGA-II, and SMS-EMOA from the Pymoo library [8], MOPSO and MOEA/D from the Pygmo library [1], and SPEA2 from jMetalPy library [7]. Variations in libraries, optimization techniques, or default settings may influence algorithm performance and comparability. To mitigate this, we selected widely used packages, reviewed documentation, conducted sensitivity analyses, and aligned parameters. However, using different packages may still introduce some uncertainty, and we recommend future studies to investigate the robustness of our findings with alternative implementations.

External Validity. The external validity of our study may be limited by the specific datasets, LLMs, and prompt templates used in our experiments. Although we aimed to select representative datasets and prompt templates, the generalizability of our findings to other datasets, domains, or LLMs warrants further investigation. To enhance external validity, we recommend future research exploring the application of schedule optimization across a broader range of datasets with diverse domains and characteristics. Additionally, assessing the proposed approach with different LLMs and prompt templates would offer further insights into its performance and generalizability. Furthermore, the quality and variety of available prompt templates may impact the effectiveness of the schedule. In scenarios where prompt templates are limited or of poor quality, the benefits of schedule optimization may be diminished. Therefore, the external validity of our findings should be considered in the context of the specific prompt templates used in this study.

7 Conclusion

This study explores the potential of combining schedule optimization with ICL to enhance LLM utilization. We formulate the problem as a multi-objective optimization task, aiming to find optimal schedules that select the most suitable prompt template for each LLM job, maximizing accuracy while minimizing cost. Our findings demonstrate the potential of this approach in improving LLM utilization, although there is still ample room for further improvement.

Our source code and complete experimental results are available at https://github.com/LLMs-EffiUse-Lab/Sched-ICL-Empirical.

Acknowledgments

This work is supported by Australian Research Council (ARC) Discovery Projects (DP200102940, DP220103044). We also thank the anonymous reviewers for their valuable and constructive feedback, which significantly improves this paper.

References

[1] 2015. Welcome to PyGMO. Retrieved April 20, 2024 from https://esa.github.io/pygmo/index.html

[2] Charu C Aggarwal and ChengXiang Zhai. 2012. A survey of text classification algorithms. *Mining text data* (2012), 163–222.

[3] Entisar S Alkayal, Nicholas R Jennings, and Maysoon F Abulkhair. 2016. Efficient task scheduling multi-objective particle swarm optimization in cloud computing. In *2016 IEEE 41st conference on local computer networks workshops (LCN workshops)*. IEEE, 17–24.

[4] AR Arunarani, Dhanabalachandran Manjula, and Vijayan Sugumaran. 2019. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems* 91 (2019), 407–415.

[5] Onur Ascigil, Argyrios G Tasiopoulos, Truong Khoa Phan, Vasilis Sourlas, Ioannis Psaras, and George Pavlou. 2021. Resource provisioning and allocation in function-as-a-service edge-clouds. *IEEE Transactions on Services Computing* 15, 4 (2021), 2410–2424.

[6] Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, and Ludovic Salomon. 2021. Performance indicators in multiobjective optimization. *European journal of operational research* 292, 2 (2021), 397–422.

[7] Antonio Benítez-Hidalgo, Antonio J. Nebro, José García-Nieto, Izaskun Oregi, and Javier Del Ser. 2019. jMetalPy: A Python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation* 51 (2019), 100598. https://doi.org/10.1016/j.swevo.2019.100598

[8] J. Blank and K. Deb. 2020. pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[10] Daniele Catanzaro, Raffaele Pesenti, and Roberto Ronco. 2023. Job scheduling under Time-of-Use energy tariffs for sustainable manufacturing: a survey. *European Journal of Operational Research* 308, 3 (2023), 1091–1109.

[11] Huangke Chen, Xiaomin Zhu, Guipeng Liu, and Witold Pedrycz. 2018. Uncertainty-aware online scheduling for real-time workflows in cloud service environment. *IEEE Transactions on Services Computing* 14, 4 (2018), 1167–1178.

[12] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176* (2023).

[13] Long Cheng, Yue Wang, Feng Cheng, Cheng Liu, Zhiming Zhao, and Ying Wang. 2023. A Deep Reinforcement Learning-Based Preemptive Approach for Cost-Aware Cloud Job Scheduling. *IEEE Transactions on Sustainable Computing* (2023), 1–12.

[14] Shuiguang Deng, Zhengzhe Xiang, Javid Taheri, Mohammad Ali Khoshkholghi, Jianwei Yin, Albert Y Zomaya, and Schahram Dustdar. 2020. Optimal application deployment in resource constrained distributed edges. *IEEE transactions on mobile computing* 20, 5 (2020), 1907–1923.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[16] Mingyue Feng, Xiao Wang, Yongjin Zhang, and Jianshi Li. 2012. Multi-objective particle swarm optimization for resource allocation in cloud computing. In *2012 IEEE 2nd international conference on cloud computing and intelligence systems*, Vol. 3. IEEE, 1161–1165.

[17] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R Lyu. 2023. What makes good in-context demonstrations for code intelligence tasks with llms?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 761–773.

[18] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. 2013. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of computer and system sciences* 79, 8 (2013), 1230–1242.

[19] Arunkumar Gopu and Neelanarayanan Venkataraman. 2019. Optimal VM placement in distributed cloud environment using MOEA/D. *Soft Computing* 23, 21 (2019), 11277–11296.

[20] Haithem Hafsi, Hamza Gharsellaoui, and Sadok Bouamama. 2022. Genetically-modified Multi-objective Particle Swarm Optimization approach for high-performance computing workflow scheduling. *Applied soft computing* 122 (2022), 108791.

[21] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[22] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620* (2023).

[23] Zhihan Jiang, Jinyang Liu, Zhuangbin Chen, Yichen Li, Junjie Huang, Yintong Huo, Pinjia He, Jiazhen Gu, and Michael R Lyu. 2023. LLMParser: A LLM-based Log Parsing Framework. *arXiv preprint arXiv:2310.01796* (2023).

[24] Qi Kang, Xinyao Song, MengChu Zhou, and Li Li. 2018. A collaborative resource allocation strategy for decomposition-based multiobjective evolutionary algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49, 12 (2018), 2416–2423.

[25] Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. Large language models are few-shot testers: Exploring llm-based general bug reproduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2312–2323.

[26] Shahriar Karami, Sadoon Azizi, and Fardin Ahmadizar. 2024. A bi-objective workflow scheduling in virtualized fog-cloud computing using NSGA-II with semi-greedy initialization. *Applied Soft Computing* 151 (2024), 111142.

[27] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering*. 1095–1106.

[28] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing: How Far Can ChatGPT Go?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE.

[29] Lei Liu, Jie Feng, Xuanyu Mu, Qingqi Pei, Dapeng Lan, and Ming Xiao. 2023. Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems* (2023).

[30] Chuan Luo, Bo Qiao, Xin Chen, Pu Zhao, Randolph Yao, Hongyu Zhang, Wei Wu, Andrew Zhou, and Qingwei Lin. 2020. Intelligent Virtual Machine Provisioning in Cloud Computing.. In *IJCAI*. 1495–1502.

[31] Ruchika Malhotra, Megha Khanna, and Rajeev R. Raje. 2017. On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions. *Swarm and Evolutionary Computation* 32 (2017), 85–109. https://doi.org/10.1016/j.swevo.2016.10.002

[32] Zhang Miao, Peng Yong, Yang Mei, Yin Quanjun, and Xie Xu. 2021. A discrete PSO-based static load balancing algorithm for distributed simulations in a cloud environment. *Future Generation Computer Systems* 115 (2021), 497–516.

[33] A Jemshia Miriam, R Saminathan, and S Chakaravarthi. 2021. Non-dominated Sorting Genetic Algorithm (NSGA-III) for effective resource allocation in cloud. *Evolutionary Intelligence* 14 (2021), 759–765.

[34] Elnaz Parvizi and Mohammad Hossein Rezvani. 2020. Utilization-aware energy-efficient virtual machine placement in cloud networks using NSGA-III meta-heuristic approach. *Cluster computing* 23, 4 (2020), 2945–2967.

[35] Kai Peng, Hualong Huang, Bohai Zhao, Alireza Jolfaei, Xiaolong Xu, and Muhammad Bilal. 2022. Intelligent computation offloading and resource allocation in IIoT with end-edge-cloud computing using NSGA-III. *IEEE Transactions on Network Science and Engineering* (2022).

[36] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[37] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[38] V Krishna Reddy, K Deva Surya, M Sai Praveen, B Lokesh, A Vishal, and K Akhil. 2016. Performance analysis of Load Balancing Algorithms in cloud computing environment. *Indian Journal of Science and Technology* (2016).

[39] Sahar Saeedi, Reihaneh Khorsand, Somaye Ghandi Bidgoli, and Mohammadreza Ramezanpour. 2020. Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing. *Computers & Industrial Engineering* 147 (2020), 106649.

[40] A Sathya Sofia and P GaneshKumar. 2018. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using NSGA-II. *Journal of Network and Systems Management* 26 (2018), 463–485.

[41] Xianpeng Wang, Hangyu Lou, Zhiming Dong, Chentao Yu, and Renquan Lu. 2023. Decomposition-based multi-objective evolutionary algorithm for virtual machine and task joint scheduling of cloud computing in data space. *Swarm and Evolutionary Computation* 77 (2023), 101230.

[42] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[43] Zhenkun Wang, Qingyan Li, Genghui Li, and Qingfu Zhang. 2023. Multi-objective decomposition evolutionary algorithm with objective modification-based dominance and external archive. *Applied Soft Computing* 149 (2023), 111006.

[44] Zhenkun Wang, Yew-Soon Ong, and Hisao Ishibuchi. 2018. On scalable multiobjective test problems with hardly dominated boundaries. *IEEE Transactions on Evolutionary Computation* 23, 2 (2018), 217–231.

[45] Bo Xu, Zhiping Peng, Fangxiong Xiao, Antonio Marcel Gates, and Jian-Ping Yu. 2015. Dynamic deployment of virtual machines in cloud computing using multi-objective optimization. *Soft computing* 19 (2015), 2265–2273.

[46] Xiaolong Xu, Shucun Fu, Weimin Li, Fei Dai, Honghao Gao, and Victor Chang. 2020. Multi-objective data placement for workflow management in cloud infrastructure using NSGA-II. *IEEE Transactions on Emerging Topics in Computational Intelligence* 4, 5 (2020), 605–615.

[47] Xiaolong Xu, Ruichao Mo, Fei Dai, Wenmin Lin, Shaohua Wan, and Wanchun Dou. 2020. Dynamic Resource Provisioning With Fault Tolerance for Data-Intensive Meteorological Workflows in Cloud. *IEEE Transactions on Industrial Informatics* 16, 9 (2020), 6172–6181. https://doi.org/10.1109/TII.2019.2959258

[48] Shuangshuang Yao, Zhiming Dong, Xianpeng Wang, and Lei Ren. 2020. A multiobjective multifactorial optimization algorithm based on decomposition and dynamic resource allocation strategy. *Information Sciences* 511 (2020), 18–35.

[49] Sonia Yassa, Rachid Chelouah, Hubert Kadima, Bertrand Granado, et al. 2013. Multi-objective approach for energy-aware workflow scheduling in cloud computing environments. *The Scientific World Journal* 2013 (2013).

[50] Tiantian Zhang, Yuliang Shi, Meng Xu, and Lizhen Cui. 2012. A Service Provisioning Strategy Based on SPEA2 for SaaS Applications in Cloud. In *2012 Second International Conference on Cloud and Green Computing*. IEEE, 290–295.

[51] Lucia Zheng, Neel Guha, Brandon R Anderson, Peter Henderson, and Daniel E Ho. 2021. When does pretraining help? assessing self-supervised learning for law and the casehold dataset of 53,000+ legal holdings. In *Proceedings of the 18th International Conference on Artificial Intelligence and Law (ICAIL)*. 159–168.

[52] Amelie Chi Zhou, Bingsheng He, Xuntao Cheng, and Chiew Tong Lau. 2017. A Declarative Optimization Engine for Resource Provisioning of Scientific Workflows in Geo-Distributed Clouds. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2017), 647–661. https://doi.org/10.1109/TPDS.2016.2599529

[53] Jiajun Zhou, Xifan Yao, Yingzi Lin, Felix TS Chan, and Yun Li. 2018. An adaptive multi-population differential artificial bee colony algorithm for many-objective service composition in cloud manufacturing. *Information Sciences* 456 (2018), 50–82.

[54] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *ICSE-SEIP*. IEEE, 121–130.