SchedMate: Large Language Models are DL Scheduling Enhancers

Zerui Wang*†
Shanghai AI Laboratory

Qinghao Hu*
NTU

Ana Klimovic
ETH Zurich

Xingcheng Zhang, Peng Sun
Shanghai AI Laboratory

Abstract

Existing deep learning cluster schedulers make their scheduling decisions on partial job information, such as resource utilization metrics obtained through profiling. Incorporating a broader range of information dimensions could enable these schedulers to make more optimal decisions. Recent advancements in Large Language Models (LLMs) present a promising avenue for efficiently extracting valuable insights from original code base and execution logs, potentially enhancing scheduling efficacy. In this work, we propose `SchedMate`, designed to enhance the performance of deep learning schedulers by incorporating LLMs and several advanced techniques. It employs an LLM-based agent to meticulously analyze user files within the working directory, extracting comprehensive metadata based on scheduling parameters. `SchedMate` utilizes a version management module to store and track the historical jobs' metadata and their fine-grained changes, enabling fast extraction of metadata for repeated jobs and lower token consumption. We plan to integrate `SchedMate` into multiple novel DL schedulers to further improve their performance.

1 Introduction

In Deep Learning (DL) clusters, schedulers play a pivotal role in managing the execution of diverse and computationally intensive jobs. Existing DL schedulers are tasked with allocating resources efficiently, yet they encounter several substantial challenges that compromise their effectiveness. They typically make their scheduling decisions on partial job information, such as resource utilization obtained via profiling. Additionally, they are usually unaware of the semantics of jobs, such as model architecture, the type of training task, batch size, and hyperparameters. However, incorporating a broader range of information dimensions could enable these schedulers to make more optimal decisions. The original code base and execution logs, for instance, inherently contain a wealth of information. Traditional approaches, however, fall short in effectively leveraging this rich data.

In light of these limitations, the recent advancements in artificial intelligence, particularly the emergence of LLMs, offer promising solutions. LLMs have revolutionized natural language processing. By leveraging the capabilities of LLM-based agents, it is possible to address the aforementioned challenges head-on. The introduction of techniques such as ReAct [6], and Retrieval-Augmented Generation (RAG) [3] provides a robust framework for LLMs to process and generate the context-rich information required for smarter scheduling decisions. This blend of AI advancements promises to significantly mitigate the current limitations of DL cluster schedulers, paving the way for more responsive, interactive, and efficient scheduling solutions.

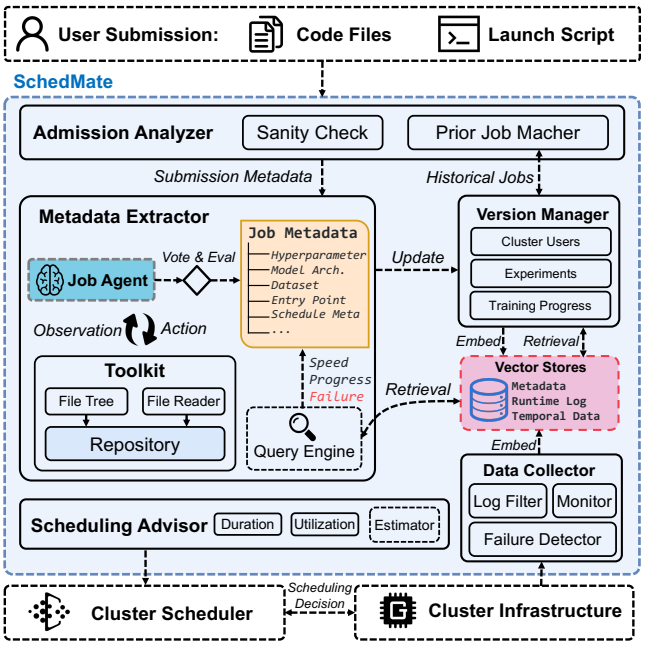

Figure 1: Overview of `SchedMate`.

In this work, we propose a novel framework, `SchedMate`, designed to enhance the efficiency of deep learning schedulers through the integration of LLMs and several advanced techniques. `SchedMate` employs an LLM-based agent to meticulously analyze user files within the working directory, extracting comprehensive metadata based on scheduling parameters. This enables `SchedMate` to intelligently predict job durations and resource needs, allowing for adaptive adjustments to the scheduling policy.

2 SchedMate Design

The core principle of `SchedMate` is to leverage LLMs to improve scheduling decisions by distilling metadata within users' repositories in a non-intrusive way. The job metadata here covers both scheduling and training-related metadata, such as the structure of the model, training hyperparameters, and dataset settings. The architecture and workflow of `SchedMate` are illustrated in Figure 1.

Upon submission, essential details of jobs are immediately

*Equal Contribution.
†Ph.D. Student

Condensed Prompt

You are a helpful AI assistant. You have access with access to several tools: `TOOLS INSTRUCTION`

ReAct Instruction. Answer questions using this process: Thought, Action, Action Input, Observation (repeat N times), and Final Answer.

Task Requirements. Find the answer based on the repository's path, command, and launch configuration. Response in a structured format.

Question.: What is the training, model, and dataset metadata of the job?

```
training command: python -m train experiment=pile/gpt
repository path: /path/to/project
launch configuration: ngpu=8, ...,nnodes=1
```

Agent Output

```
model_config       training_config    dataset_config
  model_name: GPT    batch_size: 256    train: pile
  task: NLP          lr: 0.001          valid: pile-val
  d_model: 768       optimizer: adamw   ...
  n_layer: 12        iters: 80000
  vcb_size: 50257    iter_type: step
```

Figure 2: Sample prompts and output for the *Log Agent*, with some information omitted.

processed by the *Admission Analyzer*. This includes the job name, working directory, the shell script, among others. Initially, the *Admission Analyzer* performs a rapid comparison against the user's historical submissions. Then, it forwards the basic details and any matched jobs to the *Metadata Extractor*.

Metadata Extractor is at the heart of our system, responsible for extracting the metadata from several data sources, including the source code of the job, the configuration file, the runtime log, and monitoring data.

A core component of *Metadata Extractor* is the LLM-based *Job Agent* prompted as a ReAct [6] style agent equipped with two tools, file tree reader and file reader, used for reading certain files within repositories of jobs. The *Job Agent* is fed with the job's running command and the repository's path. The agent can deftly navigate to files with required metadata using these tools. We employ a *Vote and Evaluate* strategy to mitigate the well-known hallucination issue of LLMs. Additionally, the *Query Engine*, adopting the RAG framework [3], seamlessly integrates into this system, parsing runtime logs to attain job training speed, progress, and failures.

Invoking the agent to perform a complete extraction for each task can be time-consuming. Given that users often submit similar jobs repeatedly [2, 5], we design a *Version Manager* to track users' development progress and cache historical job metadata along with their locations in the job's repository, enabling quick metadata extraction by *Metadata Extractor* for matching jobs and updates.

The *Vector Store* facilitates the embedding and retrieval of several data sources from other modules. It functions as a knowledge base, storing the runtime logs, temporal data, and the embedded representations of historical job metadata. These embeddings are generated by an embedding model. When a new job is submitted, the extracted job metadata is embedded. The *Vector Store* employs sophisticated indexing mechanisms to support efficient similar job matching and parsing of job logs from *Query Engine*.

Finally, *Scheduling Advisor* leverages machine learning models to forecast job duration and resource utilization by analyzing comprehensive metadata of current job and historical metadata. This information is then used to guide the cluster scheduler's scheduling decisions.

3 Preliminary Result and Future Plan

LLM Extraction Effect. We implement *Metadata Extractor* and conduct preliminary evaluations to demonstrate its efficiency and accuracy. The primary prompt and a sample output of *Log Agent* are shown in Figure 2. We also compare the performance of different LLMs, including GPT-4 Turbo, Qwen1.5-7B, and InternLM2-7B. Results show that they can generally follow the instructions and generate structured output. Currently, GPT-4 Turbo is the most accurate model. We will explore the effect of more LLMs (e.g., LLaMA) to comprehensively evaluate the impact of different LLMs.

Future Plan. We plan to integrate `SchedMate` into multiple novel DL schedulers as a plugin to further improve their scheduling performance. Specifically,

- **Tiresias** [1]: Making job queue duration-aware to reduce redundant and useless preemption.
- **Lucid** [2]: Achieving non-intrusive job throughput monitoring and skip profiling via prior job information.
- **Pollux** [4]: Reusing job goodput information to skip the configuration exploration stage.

References

[1] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo. Tiresias: A GPU cluster manager for distributed deep learning. NSDI '19.

[2] Q. Hu, M. Zhang, P. Sun, Y. Wen, and T. Zhang. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. ASPLOS '23.

[3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. NeurIPS '20.

[4] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. OSDI '21.

[5] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. NSDI '22.

[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. ICLR '23.