Semantic-Aware Scheduling for GPU Clusters with Large Language Models

Zerui Wang*
Shanghai Jiao Tong University &
Shanghai AI Laboratory
Shanghai, China
wangzerui@sjtu.edu.cn

Qinghao Hu*
Nanyang Technological University
Singapore, Singapore
qinghao.hu@ntu.edu.sg

Ana Klimovic
ETH Zurich
Zurich, Switzerland
aklimovic@ethz.ch

Tianwei Zhang
Nanyang Technological University
Singapore, Singapore
tianwei.zhang@ntu.edu.sg

Yonggang Wen
Nanyang Technological University
Singapore, Singapore
ygwen@ntu.edu.sg

Peng Sun
Shanghai AI Laboratory
Shanghai, China
sunpeng@pjlab.org.cn

Dahua Lin
The Chinese University of Hong Kong
Hong Kong, China
dhlin@ie.cuhk.edu.hk

Abstract

Deep learning (DL) schedulers are pivotal in optimizing resource allocation in GPU clusters, but operate with a critical limitation: they are largely blind to the semantic context of the jobs they manage. This forces them to rely on limited metadata, leading to high profiling overhead, unreliable duration estimation, inadequate failure handling, and poor observability. To this end, we propose SchedMate, a framework that bridges this semantic gap by systematically extracting deep insights from overlooked, unstructured data sources: source code, runtime logs, and historical jobs. SchedMate enhances existing schedulers non-intrusively through three LLM-based components. Our implementation integrates seamlessly with existing deep learning schedulers. Evaluations on a 128-GPU physical cluster and extensive simulations on production traces show SchedMate reduces average job completion times by up to 1.91×, substantially enhancing the scheduling performance, demonstrating the critical role of semantic-awareness in modern DL scheduling.

1 Introduction

Deep Learning (DL) has experienced unprecedented growth in recent years, with large language models (LLMs) like GPT-5 [50], LLaMA-3 [41], and GLM-4.5 [72] pushing the boundaries of AI capabilities. This growth drives a substantial demand for computational resources, particularly multi-billion dollar GPU clusters [22, 23, 28] and tailored frameworks [7, 45, 48, 65, 86]. DL schedulers are critical in orchestrating the execution of diverse and resource-intensive workloads within these clusters.

Despite extensive research on optimizing job completion time (JCT) and resource utilization [4, 17, 18, 22–24, 27, 47, 54, 78, 82, 84], existing schedulers are hampered by a fundamental **semantic gap.** They operate on surface-level metadata, leading to four significant real-world challenges:

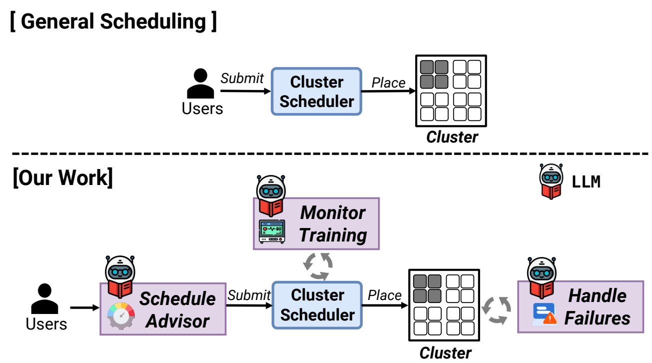

Figure 1. Existing DL schedulers operate on limited metadata (top), while SchedMate enriches the scheduling process with deep insights from source code, logs, and historical jobs (bottom).

- **C1: High Profiling Overhead.** To forecast resource needs, many schedulers [24, 27, 44, 47] rely on job profiling. This process is prohibitively expensive, especially since most production jobs are short-lived. For instance, the median job duration in the Acme cluster is just two minutes [23] (Figure 2a), meaning profiling can consume as much time as the job's entire execution. For distributed LLM training, this overhead can occupy hundreds of GPUs, imposing a significant operational burden on the cluster.
- **C2: Unreliable Duration Estimation.** Duration-aware policies [22, 24, 44, 47, 75] depend on accurate job runtime predictions, but current methods can be unreliable in production scenarios. Intrusive techniques [25, 44, 54, 66, 77] that require users to provide `num_steps` and `step_time` fail because they assume jobs run to completion. However, production data shows that up to half of all jobs terminate prematurely [22, 23, 28] (Figure 2b), often rendering user-provided estimates inaccurate. ML-based methods [24, 75]

*Both authors contributed equally to this research.

typically shows poor prediction accuracy, as they use only sparse metadata (e.g., job name, username) and lack the rich features needed for precise predictions.

- **C3: Inadequate Failure Handling.** Job failures are frequent and costly in large-scale training [23, 28, 41, 85], yet most state-of-the-art schedulers [24, 27, 47, 54] lack robust failure-handling mechanisms. For example, the pretraining of LLaMA-3.1 encountered 419 failures, averaging one every three hours [41]. This oversight leads to wasted resources and significant delays, especially for large distributed jobs where rapid recovery is essential.
- **C4: Limited Observability.** Schedulers are often blind to runtime dynamics like training progress or performance regressions. This creates a dilemma: *non-intrusive* schedulers are easy to deploy but see only surface-level metadata, while *intrusive* schedulers that modify DL frameworks to gain deeper observability are brittle and impractical. They impose a high maintenance burden [24] and cannot keep pace with the rapid evolution of frameworks [3, 52]. The intrusive scheduler Pollux, for example, is now incompatible with modern PyTorch due to years of inactivity [54]. Consequently, most production clusters [22, 23, 28] opt for non-intrusive approaches, sacrificing observability.

These challenges stem from a common root cause: the scheduler's inability to access the rich *semantic information* of a DL job. We argue that this critical information is not missing but is instead locked away in unstructured data artifacts that have been traditionally overlooked: the job's *source code* and *runtime logs*. Source code contains inherent details about workload characteristics (for **C1**, **C2**), while logs record runtime dynamics and failure messages (for **C3**, **C4**). The key research questions are **what semantic information to extract**, **how to extract it efficiently**, and **how to use it to enhance scheduling** without imposing new burdens on users or operators.

To bridge this semantic gap, we introduce SchedMate, a framework that unlocks this information to enable a new paradigm of *semantic-aware scheduling*. SchedMate integrates seamlessly into existing scheduling workflows via three LLM-powered modules (Figure 1): (1) The **Scheduling Advisor** addresses **C1** and **C2** by using an LLM agent to analyze job source code. It extracts key workload metadata to find similar historical jobs, enabling accurate workload prediction without costly profiling. (2) The **Metric Tracker** tackles **C4** with a non-intrusive, two-stage pipeline to extract training progress from verbose logs. By using efficient embedding-based filtering followed by precise LLM-based extraction, it provides real-time visibility for dynamic scheduling adjustments. (3) The **Failure Handler** resolves **C3** by performing automated root cause analysis on logs. It rapidly pinpoints error messages, identifies the cause, and can trigger automated recovery actions for infrastructure-related issues, minimizing downtime.

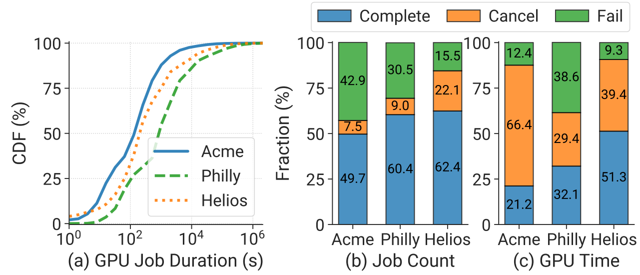

Figure 2. Background: DL workload characteristics across Microsoft Philly [28], SenseTime Helios [22] and Acme [23] clusters. (a) CDF of the job duration. (b, c) Final statuses of jobs in terms of quantity and utilized GPU resources.

We implement SchedMate as a plug-and-play module and demonstrate its effectiveness through three case studies: enhancing the non-intrusive scheduler Lucid [24], the elastic scheduler Sia [27], and as a standalone scheduler. We conduct comprehensive experiments on a 128-GPU physical cluster and extensive simulations on public production traces, from Microsoft [28], SenseTime [22], and Acme [23]. To enable fine-grained analysis, we also introduce Mars, a new trace collected from our in-house cluster containing rich source code, logs, and metrics from real-world large-scale LLM development jobs. The results show that SchedMate reduces average job completion time by up to 1.91×. These findings demonstrate that by bridging the semantic gap, SchedMate effectively addresses challenges **C1–C4** and significantly enhances the performance of modern DL schedulers.

2 Background and Motivation

2.1 DL Workload Scheduling

Existing DL Schedulers. DL jobs are usually submitted to multi-tenant GPU clusters [22, 23, 28, 75]. Figure 1 shows common paradigms of DL workload scheduling [16, 75, 83]. Schedulers manage these jobs based on policies like First-In-First-Out (FIFO) or Shortest Job First (SJF) [16]. To optimize for goals like minimizing job completion time (JCT), advanced schedulers employ techniques such as *job-packing* (co-locating jobs) [24, 78, 84], *elastic-scheduling* [27, 54, 66, 79], and *heterogeneous-aware policies* [5, 27, 47]. Some schedulers also focus on other specific like model-targeted scheduling [9, 34], inference workloads [29, 37, 69].

Workload Estimation. Effective scheduling requires estimating workload characteristics, such as duration and resource utilization. Schedulers gather necessary metrics via pre-profiling (briefly running jobs beforehand, leveraging their iterative nature [24, 47]) or online profiling (collecting metrics during execution [27, 54]). Two main duration estimation approaches are used: (1) *Config-based* methods calculate $step\_time \times total\_steps$ [14, 44, 47, 53, 54], but are unreliable under high failure/cancellation rates. (2) *History-based* methods exploit recurring job submission patterns [22,

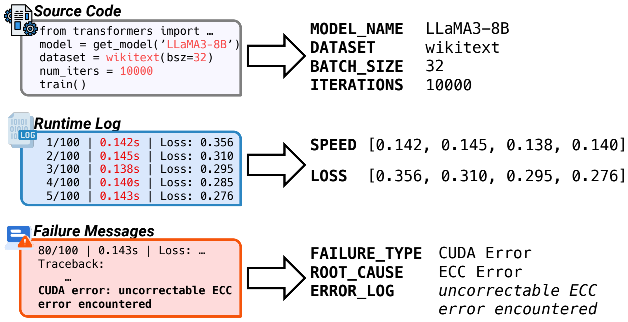

Figure 3. Examples of semantic information available in source code and runtime logs that are opaque to traditional schedulers.

23, 75]. They usually utilize the resource profile and scheduling information of historical jobs to train predictive models [22, 24, 75] or directly identify similar historical jobs [26, 51] for workload estimation.

Prevalence of Failures. Recent studies [23, 28, 30, 41] report that DL training jobs suffer from frequent failures, especially infrastructure failures. Acme [23] reports around 2 infrastructure failures per day on average during LLM development. ByteDance [30] experienced over 100 failures during a multi-week LLM pretraining task. As shown in Figure 2(c), it is obvious that only approximately 20~50% of resources are consumed by jobs that are finally complete, demonstrating the importance of failure handling.

Despite their sophistication, these schedulers operate on a thin layer of structured metadata, leaving them blind to the rich, semantic context of the workloads they manage, leading to the challenges outlined in the §1.

2.2 The Opportunity: Semantic-Aware Scheduling

We argue that the critical information needed to bridge this semantic gap already exists within the cluster but is locked away in unstructured data artifacts traditionally ignored by schedulers: a job's **source code**, its **runtime logs**, and the collective history of **all jobs**. Production traces confirm these artifacts are widely available. For instance, production GPU clusters of Microsoft [28], SenseTime [22], and Acme [23] have access to the complete source code and logs for every job. The core challenge of this paper is how to systematically extract and leverage this semantic information for enhancing the scheduling of state-of-the-art cluster schedulers.

Why Large Language Models? The information in source code and logs is unstructured and context-dependent. Traditional tools are too brittle to parse this data reliably across different DL frameworks and user code. LLMs are well-suited for this diverse task, as they can understand natural language and code structure, allowing them to perform the reasoning needed to extract workload metadata, find metrics in noisy logs, and diagnose failures. Although many recent works [35, 43] have studied how LLMs can be used to skillfully perform similar information extraction tasks, to the best of our knowledge, we are the first to harness LLM-extracted information to enhance the performance of DL schedulers.

Semantic Workload Similarity. Source code contains information of workload characteristics, available at submission time, but *how to leverage it for workload prediction, especially resource profile and duration prediction?* We find that jobs with similar workloads often exhibit similar resource profiles and duration distributions. For example, Acme [23] shows that jobs with different workload types (MLLM, pretrain, SFT) exhibit distinct duration distributions, a pattern we also observe in our Mars trace. This suggests that we can use semantic information from source code to *identify similar historical jobs*, which can then be used to predict the resource profile and duration of new jobs. Existing history-based schedulers define similarity by matching basic metadata [24, 75]. We demonstrate that a better workload similarity is semantic, which addresses challenges **C1** and **C2**.

Observability from Logs. Runtime logs offer a non-intrusive window into job progress through metrics like `step_time` and `loss`, which is ideal for production schedulers that prefer non-intrusive schedulers like Lucid [24]. It is straightforward to use LLM to non-intrusively parse logs and extract these metrics, but concerns are that logs are often verbose and may not contain the necessary information. Firstly, we can remove irrelevant logs by embedding-based filtering before LLM processing, which incurs minimal overhead compared to LLMs [8]. Furthermore, considering diverse scenarios, we can develop a DL scheduler enhancement that works when only part of the logs provides the needed information and falls back to the default behavior when the logs are insufficient. This addresses challenge **C4**.

Failure Handling with Logs. Production data shows a key distinction among job failures. Application-level bugs (e.g., syntax errors) require manual intervention. In contrast, *infrastructure failures* (e.g., network issues, hardware faults) are often recoverable by restarting the job [23, 28]. As shown in Figure 2c failed jobs consume significant GPU resources. Automatically handling these failures can significantly reduce resource waste and enhance scheduling performance. Existing schedulers do not handle failures well. We are motivated to study how failure handling enhances DL scheduling, thereby addressing challenge **C3**.

In summary, our approach is not merely about applying LLMs to logs and code, which is a well-studied area [35, 43]. Our core contribution is twofold: (1) identifying what specific semantic information is crucial for scheduling (workload similarity, runtime progress, failure type), and (2) designing mechanisms to leverage this information to facilitate semantic-aware scheduling.

3 SchedMate Overview and Design

System Overview. Figure 4 presents the architecture of SchedMate, a framework designed to enable semantic-aware scheduling. It bridges the semantic gap of DL schedulers by

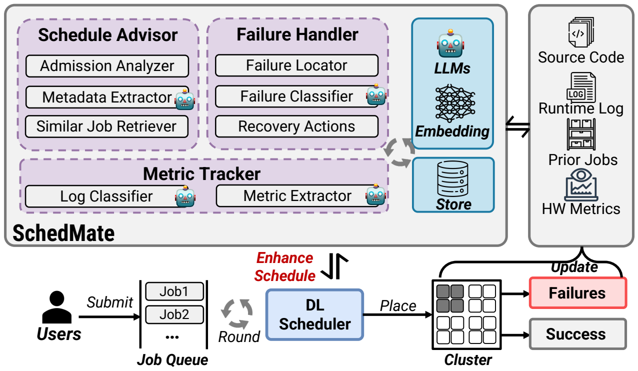

Figure 4. System Overview of SchedMate The system integrates with four data sources (prior jobs, source code, runtime log, and hardware metrics). The modules attached to robot symbols utilize LLMs.

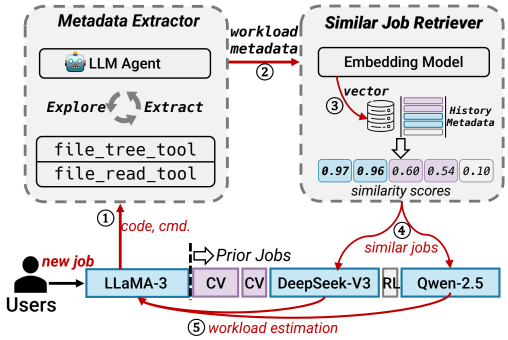

Figure 5. Scheduling Advisor Workflow. The LLM-based agent leverages two tools to support the efficient metadata extraction from source code. Length of each job refers to the duration. The red line indicates the data flow path.

extracting and integrating semantic information from four key data sources: source code, runtime logs, historical job data, and hardware metrics. SchedMate consists of three core modules: the *Scheduling Advisor*, the *Metric Tracker*, and the *Failure Handler*. The *Scheduling Advisor* analyzes source code to extract semantic workload characteristics, enabling accurate predictions without profiling. The *Metric Tracker* non-intrusively parses runtime logs to extract performance metrics, providing real-time observability into job progress. The *Failure Handler* analyzes logs to semantically diagnose job failures and automate recovery.

Semantic Workload Prediction. To replace costly profiling and improve workload estimation, the *Scheduling Advisor* uses a retrieval-based approach. The core idea is to leverage semantic information from a new job's source code to find similar historical jobs. The performance data from these past jobs is then used to predict the new job's characteristics, such as its duration and resource utilization. To efficiently extract this semantic metadata from code, we developed an LLM-based agent that reasons about the code structure to summarize key workload characteristics (in §3.1).

Progress-aware Decision Making. The *Metric Tracker* provides schedulers with real-time information of a job's runtime progress by extracting performance metrics from logs. To handle verbose and varied log formats efficiently, it employs a two-stage pipeline to efficiently filter irrelevant lines and extract structured metrics. This grants schedulers the observability needed for dynamic, progress-aware optimizations. The component's design is discussed in §3.2.

LLM-based RCA & Failure Handling. Infrastructure failures are a major source of wasted resources, yet schedulers lack the semantic context to handle them intelligently. Our *Failure Handler* provides this context through automated root cause analysis (RCA). It uses an efficient method to first locate the initial error in massive logs, then employs an LLM to classify the failure's semantic type (e.g., infrastructure vs. application). For infrastructure failures, the system can trigger automated recovery actions, minimizing manual intervention and cluster-wide resource idleness (§3.3).

3.1 Scheduling Advisor

The *Scheduling Advisor* provides workload insights by retrieving similar historical jobs, avoiding intensive profiling. It employs a retrieval-based approach that extracts metadata from a job and identifies similar historical jobs based on metadata similarity.

Semantic Workload Metadata. To capture the features of a workload, we focus on extracting key semantic metadata from the source code. Since source code can be extensive, we target the most informative details: (1) *Model Arch*, including model name, type, and task (e.g., NLP, CV); (2) *Dataset Settings*, such as dataset name; and (3) *Training Configuration*, like training steps, and hardware requirements.

Workflow. As shown in Figure 5, when a job is submitted, the *Metadata Extractor* analyzes its source code to extract this semantic metadata. The *Similar Job Retriever* then uses this metadata to find matching historical jobs. Finally, the historical performance data of these similar jobs is aggregated to produce a workload estimation for the new job, which is then used by the scheduler.

Metadata Extractor. This component is designed to extract semantic workload metadata from the job's source code. This information is often scattered across multiple files, making simple rule-based extraction brittle. To address this, we use a tool-based ReAct-style agent [81] that intelligently navigates the codebase. We equip the agent with two simple filesystem tools: `file_tree_tool`, which provides a filtered view of the project structure, and `file_read_tool`, which allows the agent to inspect file contents.

The agent's prompt is shown in Figure 6. The prompt provides the agent with: (1) the task definition, (2) instructions for using the tools (the ReAct framework), (3) the desired metadata schema, and (4) formatting instructions to ensure the output is a structured JSON. This setup enables the agent to autonomously explore the codebase, identify relevant files

Prompt for Metadata Extraction

You are a helpful AI assistant. Answer the following questions as best you can. You have access to the following tools: `TOOLS INSTRUCTION`
Task Requirements. Follow the ReAct-style instruction [81] to 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?

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

Workload Metadata Sample

```
model_config*        training_config*     dataset_config*
  model_name*: GPT     iters*: 80000        train*: pile
  task_type*: NLP      iter_type*: step     valid*: pile-val
  d_model: 768         batch_size: 256      num_workers: 4
  n_layer: 12          lr: 0.001            ...
```

Figure 6. Prompt template of *Metadata Extractor* and a workload metadata sample. Fields with * are mandatory.

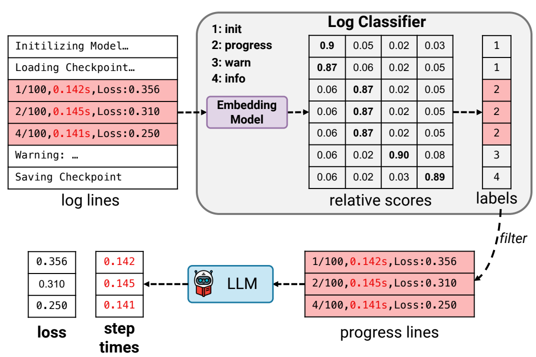

Figure 7. Metric Tracker Workflow. The progress lines are blended with some initialization and warning logs.

(like configuration scripts or model definitions), and extract the specified semantic metadata efficiently.

The agent's process, illustrated in Figure 5, is iterative. It starts with the job's working directory and command, uses the tools to explore and read files, and reasons about the gathered information to decide its next action. This continues until it has collected sufficient metadata. The final output is a structured JSON summary, providing a comprehensive semantic fingerprint of the job's workload for the retriever.

Similar Job Retrieval. To match the extracted semantic metadata with historical jobs, we use a similarity-based retrieval approach. Although the metadata has fixed fields, we still cannot directly use them for scheduling, because the values are dynamic. Therefore, instead of directly using the value of metadata for text-level operation, we use an embedding-based method. Specifically, the structured metadata is first embedded into a dense vector using an embedding model (e.g. BGE-m3 [6]). We then compute the *cosine similarity* between the new job's vector and the vectors of all historical jobs. Historical jobs with a similarity score above a threshold (T score) are considered matches. We then select the *top-k* most similar jobs to inform the workload estimation step. This semantic matching process effectively identifies jobs with truly similar workload characteristics.

Workload Estimation from Similar Jobs. Once the top-k similar historical jobs are identified, their performance data is used to predict the new job's characteristics. Specifically, we calculate the average duration and average resource utilization (e.g., SM utilization) of these similar jobs. This average value serves as the final estimation for the new job, providing a reliable forecast rooted in the performance of semantically similar past workloads. This estimation is then passed to the scheduler to inform its decisions, such as job ordering or packing. We also showcase another method of using similar jobs for performance modeling in §5.3.

3.2 Metric Tracker

The *Metric Tracker* offers runtime observability, providing schedulers with real-time insights into job performance without intrusive code modifications. But this information is buried within thousands of unstructured log lines. Directly applying LLMs to parse entire log streams is computationally prohibitive and introduces unacceptable latency for real-time scheduling. To solve this, we designed a lightweight, two-stage pipeline: a fast classifier first filters for relevant lines, and a more powerful LLM then parses only this small subset. This design achieves high-fidelity metric extraction with the low latency required for dynamic scheduling decisions. The workflow is shown in Figure 7.

Log Classifier. Production DL logs are highly heterogeneous [23]. To handle this diversity, the *Log Classifier* first categorizes each log line into high-level semantic types, as shown in Figure 7. The classification is performed by computing the cosine similarity between a line's embedding vector and category vectors. These category vectors are generated once by embedding descriptive text labels and generalize well across different logging styles. This step acts as a fast, semantic filter, isolating the small subset of lines likely to contain progress metrics.

Metric Extractor. Once the *Log Classifier* identifies progress-related lines, the *Metric Extractor* uses an LLM to parse them and extract structured data (e.g., step time, loss). To ensure low latency, we process logs in reverse chronological order and stop after successfully extracting metrics from a small number (N) of lines, which is typically sufficient for a stable estimate. To ensure robustness, we also further remove outliers. This two-stage design is highly cost-effective, as it leverages the much higher throughput of embedding models for the initial bulk filtering, reserving the more powerful but slower LLM for the final, precise extraction task.

Robustness to Partial Information. The effectiveness of the *Metric Tracker* depends on jobs logging their training progress. If a job does not output this information, the tracker

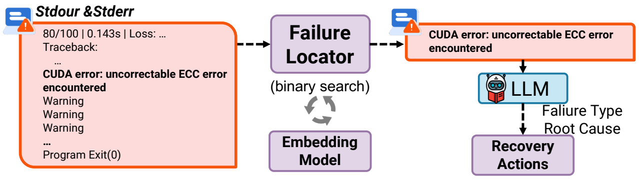

Figure 8. Failure Handler Workflow. The emphasized log line is the first failure we aims to locate.

cannot provide metrics. Our integrations into existing schedulers are designed to be robust to this scenario: if needed metrics cannot be extracted for a particular job, the scheduler falls back to its default behavior for that job, ensuring no degradation in scheduling performance.

3.3 Failure Handler

The *Failure Handler* is designed to close the semantic gap in failure management, a major source of wasted resources in GPU clusters (§2). Its goal is to empower the scheduler to distinguish between recoverable *infrastructure failures* (e.g., faulty hardware) and non-recoverable *application-level errors* (e.g., code bugs). The core technical challenge is locating the single root-cause message within massive logs that are often filled with thousands of cascading, secondary error messages. To solve this efficiently, the *Failure Handler* employs a three-stage pipeline: fast failure message localization, LLM-based classification, and automated recovery.

Failure Locator. To efficiently find the root cause in a haystack of error messages, the *Failure Locator* uses a binary search algorithm guided by our semantic *Log Classifier*. This approach is based on the empirical observation that the first true error message typically triggers a cascade of subsequent errors. This creates a semi-sorted log structure (normal logs → error logs), which is ideal for binary search. The algorithm repeatedly splits the log and uses the *Log Classifier* to check the midpoint. To improve robustness against normal logs that may be interleaved with errors after a fault, the search operates on log *chunks* rather than single lines. A chunk is classified as containing an "error" if it includes any failure-related messages, allowing the search to reliably converge on the initial failure.

Failure Classification. Once the initial error message is located, a small context window (e.g., 500 lines) around that point is passed to an LLM for fine-grained classification. The failure is categorized based on a taxonomy adapted from production cluster analysis [23], distinguishing between infrastructure failures, framework errors, and user script bugs. For infrastructure failures, the LLM performs a second classification step to identify the likely faulty component (e.g., GPU, NVLink, Node). To ensure structured output with two fields: *Error Type* and *Faulty Component*.

Automated Recovery Actions. Once the failure is classified as infrastructure-related and a faulty component is identified, SchedMate autonomously executes a predefined recovery action. For example, if a GPU failure is diagnosed, the system can: (1) run a diagnostic tool like `nccltest` to confirm the fault, (2) isolate the node in the cluster manager to prevent future placements, (3) provision a replacement node, and (4) restart the job from its last checkpoint on the new resources. This automated remediation pipeline eliminates the need for manual intervention for a common and costly class of failures, significantly reducing job downtime.

4 Integration Case Studies

4.1 Case Study 1: Non-intrusive Scheduler Lucid

Lucid [24] is a non-intrusive scheduler that uses job packing to minimize Job Completion Times (JCTs), guided by metrics from a brief, initial profiling period (T_{prof}). While effective, this design introduces several challenges in production environments.

In practice, however, this design faces several challenges that limit its effectiveness. First, the brief profiling can be inaccurate. If T_{prof} occurs during a job's initial "warmup" stage, it will underestimate the job's true steady-state resource utilization [36]. This frequently leads to suboptimal packing decisions, such as co-locating two resource-intensive jobs, which causes significant interference and performance degradation. Second, Lucid's duration estimator relies on limited, surface-level metadata (e.g., username, job name), which is insufficient for capturing the true workload characteristics of a job. Finally, Lucid lacks any mechanism to detect or react to these packing-induced slowdowns at runtime, meaning poor decisions cannot be corrected.

We integrate SchedMate to address these specific limitations with three enhancements:

Bypass Profiling. To overcome inaccurate profiling, we use the *Scheduling Advisor*. For a new job, the Advisor retrieves similar historical jobs. If a match is found ($k > 0$), we use the average of their historical hardware metrics as a more reliable prediction, bypassing the default profiler. If no match is found ($k = 0$), the system falls back to default profiling.

Enhanced Duration Estimation. We replace Lucid's limited model with the *Scheduling Advisor*'s semantic, retrieval-based approach. We use the average duration of the matched similar jobs as the new prediction. If no match is found, we fall back to Lucid's original estimator.

Interference-aware Packing Cancelation. To provide the missing feedback loop of over-slowdown, we introduce an non-intrusive interference-aware packing cancelation mechanism using the *Metric Tracker*. We take two jobs, A and B, for example. Initially, only Job A is running in the cluster. The *Metric Tracker* non-intrusively process Job A's log and obtain TP_{before}, the throughput of job A before packing is conducted. Then, Job B is scheduled and packed with Job A. After a period of time, which ensures passing the warmupstage, we then collect the newly generated log of Job A and get TP_{after}, the throughput after packing. We require that Job A has completed several tens of training steps after

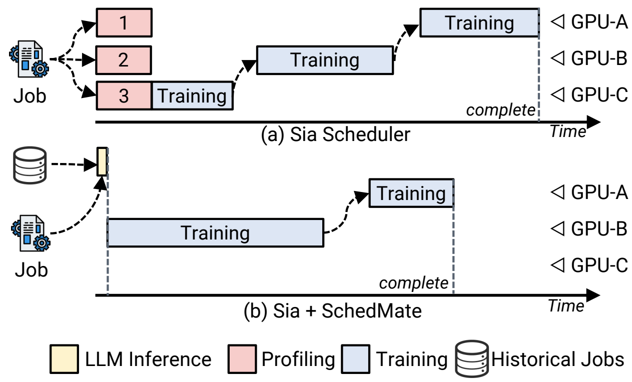

Figure 9. Job lifecycle in Sia and Sia+SchedMate. The yellow block refers to the processing of *Scheduling Advisor*. GPU-{A,B,C} represents different types of GPUs.

packing before the second triggering of *Metric Tracker* to unsure steady running of the two packed jobs. Finally, if we detect that the slowdown rate ($\frac{TP_{after}}{TP_{before}}$) of Job A is lower than a threshold (35% slowdown in the later experiments), the scheduler intelligently cancels the packing by evicting Job B and returns it back to the job queue for rescheduling. Considering that the model training progress of the evicted jobs will be lost, we restrict that the evicted jobs cannot be packed again to avoid frequent eviction. Note that the eviction check is skipped for the specific job pairs where the job's progress cannot be tracked.

4.2 Case Study 2: Elastic Scheduler Sia

Sia [27] is a state-of-the-art heterogeneity-aware, elastic scheduler designed to improve JCTs and resource utilization in heterogeneous clusters. Sia begins each job by profiling it and then adopts a profile-as-you-go approach, using this information to make scheduling decisions. It records all throughput information under all placements and batch sizes encountered by each job. However, if the initial profiling process is conducted for every job, it can be both costly and time-consuming. To address this limitation, we have integrated SchedMate into Sia, minimizing the profiling overhead by identifying resubmissions or similar jobs and bypassing their profiling process.

Pre-profiling-free Bootstrapping. We modify Sia's initial workflow. When a new job arrives, instead of immediately profiling it on GPUs, we first use the *Scheduling Advisor* to retrieve similar historical jobs. If similar jobs are found, we reuse the historical performance model from the matched jobs to bootstrap the new job, allowing it to start training immediately without any profiling. If no match is found, we profile the job in the original way. Once the job is scheduled, we continue to apply Sia's profile-as-you-go method, updating the job's throughput information. Figure 9 illustrates this process. Vanilla Sia scheduler requires profiling on each GPU type before training, consuming additional resources.

Table 1. Physical Evaluation: Comparison of policy performance between physical and simulated environments.

Policy	Cluster	Avg. JCT	Avg. Queuing
Lucid+SchedMate	Physical	**5.53h**	**2.71h**
	Simulation	5.64h	2.60h
Lucid	Physical	6.85h	3.40h
	Simulation	6.79h	3.32h

In contrast, our system leverages historical job data to bypass profiling, allowing training to start immediately and taking advantage of previous profiling results, which reduces overhead and improves cluster efficiency.

4.3 Case Study 3: Standalone Scheduler SchedMate

We implement SchedMate as a standalone scheduler following the vanilla Shortest Job First (SJF) policy. In this configuration, we employ both the *Scheduling Advisor* and *Failure Handler* modules. The standalone SchedMate follows the vanilla Shortest Job First (SJF) policy. The scheduler maintains a database to store the scheduling metadata (submit time, end time, etc.) and workload fingerprint, extracted by Metadata Extractor, of historical jobs.

For each job, the *Scheduling Advisor* retrieves three most similar historical jobs and calculates their average duration to estimate the new job's runtime. The scheduler then sorts the jobs based on these estimated durations and schedules them accordingly. In the event of a job failure, the *Failure Handler* is triggered to analyze the root cause of the failure, take actions to mitigate the issue, and resume the job. This implementation allows us to evaluate SchedMate's core capabilities in job duration estimation, efficient scheduling, and failure management independently of existing schedulers.

5 Evaluation

This section presents our evaluation of SchedMate. We begin with experiments on a physical cluster (§5.2). We then assess SchedMate's effectiveness in enhancing existing DL schedulers (§5.3, §5.3) and as a standalone scheduler (§5.3).

5.1 Experiment Setup

Implementation. We implement SchedMate on top of a Ray [46]-based scheduling system with approximately 5,500 lines of code. SchedMate is implemented as a library and can be integrated into existing schedulers. We deploy the core LLM as an API service using vLLM [33]. For the embedding models, we use the FlagEmbedding library [6]. We use Redis [62] for storing the vectors, historical job metadata, and performing similarity searching. The system incorporates real-time GPU monitoring using NVIDIA Management Library [2], providing hardware metrics for profiling.

Testbed and Models. We conduct our physical experiments on 8 nodes in a SLURM [83] cluster. Each node is equipped with 8 NVIDIA A800-80GB GPUs and 2TB of memory. For software, we use Python 3.9, CUDA 12.2, PyTorch 2.4.0, RAY

Table 2. DL model training workloads for scheduler evaluation on the physical cluster, categorized by scale.

Scale	Model	Dataset	Batch Size	Task	#Param.
Large (>1B)	LLaMA-3 [41]	Alpaca [71]	1~4	Instruction Fine-tuning	8B
	Qwen-2.5 [80]	Alpaca [71]	1~4	Instruction Fine-tuning	14B
Medium (100M - 1B)	ViT-L/16 [13]	ImageNet-1k [10]	16~64	Image Classification	307M
	BERT-Base [12]	SQuAD v1.1 [57]	16~64	Fine-tuning	110M
	DeBERTa-Large [20]	SQuAD v1.1	16~64	NLU Fine-tuning	400M
	Stable Diffusion [59]	LAION [63]	1~8	Text-to-Image	860M
	CLIP ViT-L/14 [55]	WIT [68]	16~64	Image Classification	427M
	DLRM [49]	Criteo TB [11]	512~2048	Recommendation	~180M
Small (<100M)	ResNet50 [19]	ImageNet-1k [10]	32~128	Image Classification	25.6M
	Unet2D [60]	BraTS [67]	8~64	Image Segmentation	30M
	EfficientNet-B4 [70]	ImageNet-1k [10]	32~128	Image Classification	19M
	ConvNeXt-Base [40]	ImageNet-1k [10]	32~128	Image Classification	88M
	MobileNetV2 [61]	ImageNet-1k [10]	32~128	Image Classification	3.5M
	YOLOv8 [31]	COCO [39]	16~64	Object Detection	25.9M
	T5-small [56]	C4 [56]	1~8	Text Generation	60M

2.6.3, and vLLM 0.5.5 [33]. In this section, if not specified, we use BGE-m3 [6] embedding model and Qwen-2.5-7B-Instruct [73] (enable GPTQ [15] Int8 quantization) as the core models. Additionally, we deploy the embedding model on CPU and LLM on a single GPU using vLLM. We compare the efficiency and accuracy of different models in §5.5.

Traces. We use the following traces from various sources for comprehensive evaluation of SchedMate:

- *Mars*: A production trace we collect from a 1,024 GPU cluster, containing 500 LLM training jobs (7B-100B parameters) with code, logs, and hardware metrics. These jobs utilize the same codebase with about 40K LoC.
- *Public Traces*: Widely-used public DL cluster traces including Acme [23], Philly [28], Helios [22] (Saturn and Venus), and Sia-trace [27]. Used for end-to-end physical and simulator evaluations.
- *Synthetic Traces*: We generated 100 different jobs from 20 open-source GitHub [1] repositories covering diverse tasks and models in domains, such as CV, NLP, and audio. These repositories reflect varied development practices (e.g., code organization, logging, frameworks).

Workloads. In the physical experiments, we use the models, datasets, and batch sizes in Table 2. In the simulation experiments of standalone SchedMate (§5.3), we use a sampled one-month trace from Acme without changing the workload, mainly LLM training jobs. In the Lucid (§5.3) and Sia(§5.3) simulation experiments, we follow the workload recipes of the original papers. To simulate real-world development practices, we set a portion of the job to exit early.

5.2 Evaluation on a Physical Cluster

We evaluate Lucid and Lucid+SchedMate (settings in §4.1) on a physical cluster comprising 8 nodes, each equipped with 8 NVIDIA A800-80GB GPUs (128 GPUs total). Our implementation extends the RAY [46] scheduler to incorporate Lucid and SchedMate policies, utilizing RAY's fractional GPU capabilities for job packing. Each job is submitted as a RAY task. This approach allows deployment on existing clusters without altering the underlying resource manager.

The evaluation workload consists of 241 jobs sampled from Philly and Acme. We assign workloads from Table 2 based on the job's original GPU request size: *large* (8 GPUs), *medium* (4-8 GPUs), and *small* (<4 GPUs). Job requests exceeding the cluster capacity are capped at 128 GPUs. The slowdown threshold is set to 0.5. We repeat 3 times to eliminate randomness. Lucid's profiler reserves one node and profiles each job for 100s. SchedMate operates as a RAY actor on a dedicated GPU, where we deploy Qwen-2.5-7B-Instruct and BGE-m3 as core models. Results in Table 1 show that Lucid+SchedMate reduces the average JCT by 23.2% compared to Lucid on the physical cluster. This improvement comes from SchedMate's ability to cancel subpar packing decisions and enhanced duration estimation. We also observe that the average latency of *Scheduling Advisor* is 9s, which is acceptable compared to the profiling overhead.

We also verify the fidelity of our simulator. We use the simulator in 5.3 to process the same traces and settings and compare the result with the ground truth result. The average error rate of both average JCT and makespan is less than 3.7%, which indicates the high fidelity of our simulator.

5.3 Simulation-based Evaluation

Case Study 1: Non-intrusive scheduling with Lucid. We conduct an end-to-end simulation on the performance of Lucid+SchedMate, Lucid, and Horus, QSSF, on four traces: Acme, Philly, Saturn, and Venus. Figure 10(a,b,c) presents the CDF curves of JCTs on different traces. Lucid+SchedMate consistently outperforms other policies. Figure 10(d) presents the average JCT improvements of Lucid+SchedMate against Lucid. Lucid+SchedMate demonstrates superior performance, improving 1.23× ~ 1.91× compared to Lucid alone. The key

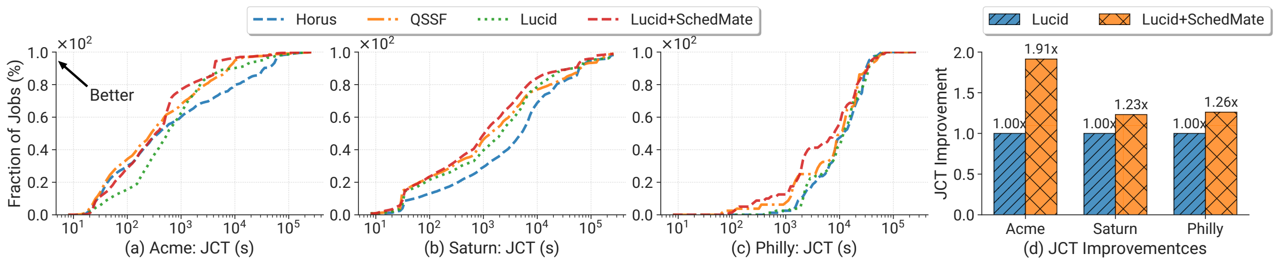

Figure 10. Case Study 1. The CDF curves of JCTs using different scheduling policies (a,b,c) and (d) Avg. JCT improvement of Lucid+SchedMate against Lucid on four traces: Acme, Philly, Saturn, and Venus.

Table 3. Case Study 2. Comparison of the performance of Sia and Sia+SchedMate across different traces.

Trace	Policy	Avg. JCT	p99 JCT	Makespan
Philly	Sia	0.74h	10.60h	14.7h
	Sia+SchedMate	**0.59h**	**10.12h**	**14.4h**
Helios	Sia	0.83h	12.0h	15.7h
	Sia+SchedMate	**0.72h**	**10.7h**	**14.8h**
Sia-trace	Sia	0.64h	4.25h	12.5h
	Sia+SchedMate	**0.53h**	**4.12h**	**12.4h**

source of improvements is its ability to dynamically detect slowdowns of job packing and predict workload using *Workload Metadata*. Specifically, among the traces, the performance gain of SchedMate is more significant on the Acme trace, owing to the fact that Acme trace contains heavier workloads, which leads to a significant slowdown in job packing. The interference-aware eviction policy of SchedMate is more effective in such traces. Besides, Acme has a high-skewed workload distribution [23], where most of the jobs are short-term jobs, introducing challenges to duration estimation. In addition, the improvements in other traces are smaller because the duration distribution is more concentrated in Saturn and Philly, and the jobs are relatively lightweight. Many short jobs are finished during profiling. The interference of jobs is also rather lower, leaving less room for the improvement of the eviction policy.

Case Study 2: Elastic Scheduling with Sia. As mentioned in §4.2, Sia+SchedMate can reduce the profiling overhead through *Preprofiling-free Bootstraping*. To test the effectiveness of SchedMate in elastic scheduling scenarios, we integrate SchedMate into Sia's simulator and conduct evaluation using identical workload and heterogeneous cluster settings in Sia's paper [27]. Table 3 presents the results. We observe that our system consistently outperforms Sia. Specifically, SchedMate reduces the average JCT by 13.3% $\sim$ 20%. Furthermore, the p99 JCT and makespans also show consistent improvements. The improvements derive from the benefits of *Scheduling Advisor* in mitigating profiling overhead while maintaining the accuracy of the scheduling decisions. In summary, this case study demonstrates that integrating SchedMate with Sia (**Sia+SchedMate**) significantly enhances scheduling efficiency.

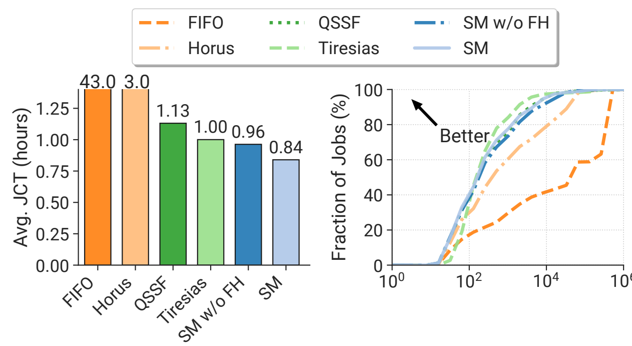

Figure 11. Case Study 3: Standalone SchedMate Evaluation. (Left) Job Completion Times (JCTs) and queuing delays. (Right) CDF curves of JCTs using different policies. **SM**: SchedMate. **FH**: *Failure Handler*.

Case Study 3: Standalone Scheduling. We evaluated the performance of SchedMate as a standalone scheduler, which utilizes *Scheduling Advisor* for duration estimation and *Failure Handler* for automatic failure recovery. We sample jobs of Acme between June and August 2023. Specifically, to evaluate the failure recovery performance, we identify resumed failed jobs from the trace and consider failures in the simulator, which is the first simulator considering this factor. We compared standalone SchedMate against several baseline schedulers: Tiresias [18] (a preemptive scheduler), Quasi-Shortest-Service-First (QSSF) [22] (ML to prioritize short jobs), FIFO, and Horus [82]. Figure 11 shows that standalone SchedMate achieves a 25.7% improvement in average JCT over QSSF and 16% over Tiresias, primarily due to its superior duration estimation capabilities provided by the *Scheduling Advisor*. Furthermore, incorporating the *Failure Handler* yields an additional ~12.5% reduction in average JCT compared to SchedMate without failure handling, highlighting its effectiveness in mitigating the impact of job failures. Overall, this case study demonstrates that standalone SchedMate significantly enhances scheduling efficiency through LLM-based workload prediction and failure management.

5.4 Module-Specific Performance Analysis

Scheduling Advisor Evaluation. We compare the performance of workload estimation of *Scheduling Advisor* against an ML-based estimator from Lucid. We use the Mars trace

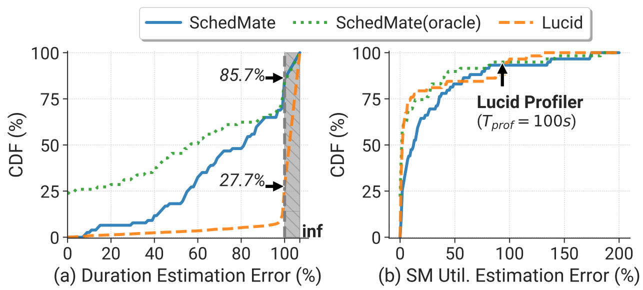

Figure 12. CDF curves of relative error for (a) duration estimation and (b) SM utilization estimation of SCHEDMATE, oracle SCHEDMATE, and Lucid.

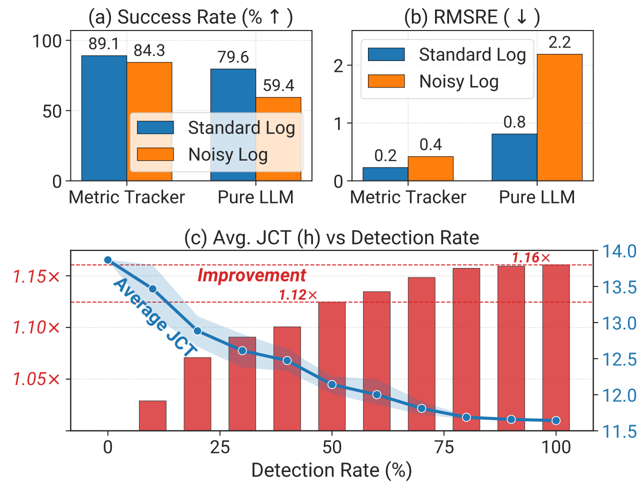

Figure 13. Metric Tracker Evaluation. Performance comparison between *Metric Tracker* and a pure LLM under standard and noisy conditions. (a) Success rate evaluation. (b) Rmsre comparision. (c) Impact of slowdown detection rate on average JCT (red) and its improvement (blue) in Lucid+SCHEDMATE (each repeated five times).

to evaluate both methods. Specifically, the Lucid estimator trains an GA^2M model [24] for duration estimation. Figure 12 illustrates the results across SCHEDMATE, Lucid, and oracle SCHEDMATE, which always gets the correct similar jobs from historical jobs. Due to the sensitivity of relative error when ground truth values are small, we truncate errors in (a) and (b). For duration estimation, SCHEDMATE achieves relative errors less than 100% for 85.7% of estimations, significantly outperforming Lucid at 27.7%. Notably, the curve shows a sharp increase as the relative error approaches 100%. Regarding SM utilization estimation, SCHEDMATE performs slightly below Lucid's profiler, with average relative errors of 28.1% and 19.3%, respectively. However, SCHEDMATE's method requires less than 10 seconds on a single GPU per job, whereas Lucid demands profiling per job for a period of time (T_{prof} = 100s in this case). Furthermore, oracle SCHEDMATE presents a promising performance in both tasks, demonstrating the potential of our retrieval-based method.

Table 4. Failure Handler Evaluation. We evaluate the component in identifying infrastructure failures against baselines. Failure Handler utilizes Qwen-2.5 models.

Method	Model	F1-score	Precision	Accuracy
Failure Parser	7B	**67.7**	**75.7**	**90.1**
	14B	**65.1**	**77.1**	**90.0**
	32B	68.2	78.4	90.7
Failure Parser w/o Locator	7B	11.1	75.0	83.8
	14B	62.5	47.9	81.8
	32B	**69.0**	**81.1**	**90.9**
RCACopilot [8]	FastText	43.0	50.0	87.0

Failure Handler Evaluation. We evaluate our failure handler on 300 failed jobs sampled from Mars, among which 75 are infrastructure failures. We set *failure handler w/o locator* as the baseline, which simply parses the last 500 log lines without attempting to locate the failure message, relying on the assumption that the relevant information is likely to be near the end of the log. For the *failure handler*, we parse the 200 log lines around the located failure message. Results are summarized in Table 4. We observe that the *Failure Handler* method consistently outperforms or matches the baseline across different model sizes, suggesting reliable identification of infrastructure failures with minimal false positives. While the baseline only shows competitive performance with the 32B model, our method achieves comparable results with smaller models, indicating better efficiency. Specifically, the 7B model achieves a comparable F1-score against the 32B model. This demonstrates that the failure locator effectively narrows down the relevant log lines, enabling smaller models to perform well without sacrificing accuracy, thereby reducing both latency and resource consumption.

We also compare our system against RCACopilot [8], a recent embedding-based approach. It achieves only 43.0 F1-score. We believe this is due to the complex failure patterns, which are not well captured by the embedding model.

Metric Tracker Evaluation. The *Metric Tracker* resolves the noisy log challenges in real-world scenarios. We tested it on a synthetic dataset of 1,000 logs, with 50% containing massive irrelevant data. We set the pure LLM method as the baseline, which processes the log without filtering. We perform *step time extraction* using both methods. As shown in Figure 13, *Metric Tracker* achieved an 84.3% success rate and an RMSRE of 0.42, significantly outperforming the pure LLM approach, whose success rate drops to 59.2%. These results highlight the *Metric Tracker*'s superior resilience to noisy log data. Besides, we also observe that our method has a lower latency with **8.7**s per log compared to 9.8s per log in pure LLM. These results demonstrate both the efficiency of the *Metric Tracker* and robustness in handling noisy data, which is common in real-world DL job logs.

5.5 Micro-benchmarks

Ablation Study on Partial Information. The non-intrusiveness of SCHEDMATE indicates the absence of certain information. We perform an ablation study on the Lucid+SCHEDMATE's

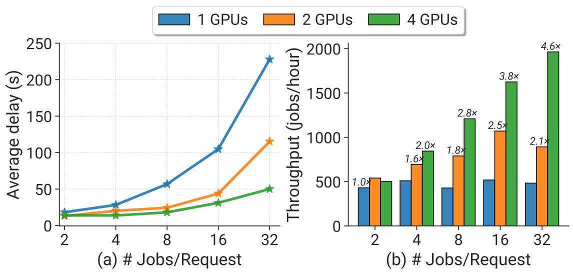

Figure 14. Performance scaling with job intensity and GPU count of the *Scheduling Advisor* using a Qwen-1.5-14B-Int8 model (a) Average delay vs. jobs per request. (b) Throughput and improvement ratios for different numbers of GPUs.

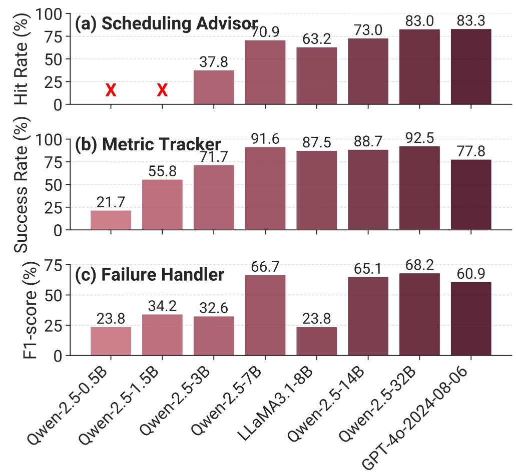

Figure 15. Evaluation on different LLMs. We involve 7 open-source models and one closed-source model, GPT-4o. The red cross mark means that the model fails in the task.

packing eviction technique. In practice, we can only detect the slowdown of jobs when they actually output the step times in their logs. To simulate different levels of partial information, we manually control the **detection rate** from 0 to 1.0. The detection rate refers to the percentage of detected slowdown jobs. As shown in Figure 13, the JCTs of Lucid on a sampled **Acme** trace with interference-aware evicting under different slowdown detection rates are presented. We assign the workloads in Table 2 and set the slowdown threshold to 0.50. The JCTs under different settings are presented in Figure 13(c). We observe that the average JCT decreases as the detection rate increases. The improvement achieves 12.5% when the detection rate is 0.5. When the detection rate is above 0.6, the performance gain is limited. *Metric Tracker* relies on the assumption that most jobs would output the metrics in the log. In addition, the result in Figure 13 shows *Metric Tracker* achieves a 91.6% success rate using a Qwen-2.5-7B model. In summary, relying on users' logs to detect the slowdown of jobs seems to be a risky approach. However, we prove that in real-world scenarios, partial information can achieve considerable performance gain.

Impact of Different LLMs. Figure 15 presents the performance of various LLMs across different scales for the three modules in SchedMate. We observe that model performance improves with increased model size, with Qwen2.5-32B achieving the best overall results among open-source models. Smaller models (0.5B and 1.5B) failed to perform the SA task, indicating that a certain level of model complexity is required for effective metadata extraction. Interestingly, the closed-source GPT-4o model demonstrates superior performance in the SA task but unexpectedly underperforms in MT and FH tasks compared to larger Qwen models. In addition, despite competitive performance in MT tasks, LLaMA-3.1-8B struggles significantly in the FH task and does not match the same scale Qwen models in the other two tasks. The results highlight a trade-off between model size, task performance, and computational requirements, with models in the 7B to 14B parameter range offering a good balance of effectiveness and efficiency for SchedMate's tasks.

Scalability and Overhead We sample 200 jobs from *Mars* and test the *Scheduling Advisor*, which takes the heaviest load among the three modules, under varying GPU counts and request intensity. We deploy Qwen-1.5-14B-Int8 model on each GPU. Note that for testing scalability on a heavier workload, we only sample large-scale jobs that use the same project with over 40K LoC.

Figure 14 (a) shows that when the request intensity is less than 4, the Avg. delay remains less than 20s, acceptable for one GPU and no significant benefit for more. When request intensity ≥ 8, delay of one GPU increases significantly, while using more GPUs maintains lower delays. Figure 14 (b) illustrates the throughput improvements. We set the *1GPU2job throughput* as the baseline and mark the throughput improvement. It's obvious that single GPU throughput remains steady (~480 jobs/hour), while the 2-GPU and 4-GPU settings increase. Specifically, the 2-GPU setting achieves over 2× throughput when request intensity reaches 16, and 4-GPU up to 4.6× (~2000 jobs/hour) throughput over the baseline setting when 32 jobs per request. In summary, we also recommend large-scale cluster operators to dynamically assign GPUs according to the actual submission rate. For instance, for PAI [75] with periodical load (300 $\sim$ 1200 jobs/h), while a single GPU is suitable for low load periods, to manage peak load periods, one should allocate 4 or more GPUs.

6 Discussion and Related Work

Supporting Other Workloads. While SchedMate targets DL training, its core principle of semantic-aware scheduling is applicable to other domains like big data analytics [58] and serverless computing [64]. The *Scheduling Advisor*, for instance, could be adapted to extract domain-specific semantics to address challenges such as data locality for big data jobs or dynamic scaling for serverless functions. Future work could explore these adaptations and integrate more advanced

LLM agent techniques [42] to more intelligently extract and utilize semantic information.

Related Work. Existing DL schedulers [4, 17, 18, 22–24, 27, 47, 54, 78, 82, 84] make efforts to optimize resource allocation and job scheduling but works without enough semantic information of jobs. Machine learning has been increasingly used to optimize systems [21, 24, 32, 76]. More recently, LLMs have been applied to specific system tasks like log parsing [35, 43] and failure diagnosis [8, 38, 74]. However, no works have been proposed to extract deep semantic information of cluster jobs to enhance DL job scheduling decisions. SchedMate is distinct in its holistic approach: it harnesses LLMs to bridge the semantic gap between DL jobs and the cluster scheduler, enabling a new paradigm of semantic-aware DL cluster scheduling.

7 Conclusion

SchedMate introduces a new paradigm of semantic-aware scheduling that extracts semantic insights of jobs using LLMs. Our evaluations show that it enhances state-of-the-art DL schedulers by improving workload prediction, enabling dynamic runtime intervention, and automating failure recovery to significantly boost cluster performance and efficiency.

References

[1] Github. https://github.com, 2023.

[2] Nvidia management library (nvml). https://developer.nvidia.com/management-library-nvml, 2024. Accessed: 2024-10-16.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '16, pages 265–283. USENIX Association, 2016.

[4] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. Online evolutionary batch size orchestration for scheduling deep learning workloads in gpu clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021.

[5] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20. Association for Computing Machinery, 2020.

[6] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation, June 2024. arXiv:2402.03216 [cs].

[7] Qiaoling Chen, Diandian Gu, Guoteng Wang, Xun Chen, YingTong Xiong, Ting Huang, Qinghao Hu, Xin Jin, Yonggang Wen, Tianwei Zhang, and Peng Sun. Internevo: Efficient long-sequence large language model training via hybrid parallelism and redundant sharding, 2024.

[8] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the 19th EuroSys Conference*, EuroSys '24. Association for Computing Machinery, 2024.

[9] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing. In *2022 USENIX Annual Technical Conference*, USENIX ATC '22, pages 199–216. USENIX Association, 2022.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[11] Aditya Desai and Anshumali Shrivastava. The trade-offs of model size in large recommendation models : A 10000 × compressed criteo-tb dlrm model (100 gb parameters to mere 10mb). *CoRR*, abs/2207.10731, 2022.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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*, NAACL '19, 2019.

[13] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[14] Abdullah Bin Faisal, Noah Martin, Hafiz Mohsin Bashir, Swaminathan Lamelas, and Fahad R. Dogar. When will my ML job finish? toward providing completion time estimates through Predictability-Centric scheduling. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 487–505, Santa Clara, CA, July 2024. USENIX Association.

[15] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[16] Wei Gao, Qinghao Hu, Zhisheng Ye, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision. *CoRR*, abs/2205.11913, 2022.

[17] Wei Gao, Xu Zhang, Shan Huang, Shangwei Guo, Peng Sun, Yonggang Wen, and Tianwei Zhang. Autosched: An adaptive self-configured framework for scheduling deep learning training workloads. In *Proceedings of the 38th ACM International Conference on Supercomputing*, ICS '24, page 473–484, New York, NY, USA, 2024. Association for Computing Machinery.

[18] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '19, pages 485–500. USENIX Association, 2019.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, 2016.

[20] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2021.

[21] Qinghao Hu, Harsha Nori, Peng Sun, Yonggang Wen, and Tianwei Zhang. Primo: Practical learning-augmented systems with interpretable models. In *2022 USENIX Annual Technical Conference*, USENIX ATC '22. USENIX Association, 2022.

[22] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021.

[23] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo,

Yonggang Wen, and Tianwei Zhang. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 709–729, 2024.

[24] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '23. Association for Computing Machinery, 2023.

[25] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic resource sharing for distributed deep learning. In *18th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '21. USENIX Association, 2021.

[26] Akshay Jajoo, Y. Charlie Hu, Xiaojun Lin, and Nan Deng. A case for task sampling based learning for cluster job scheduling. In *19th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '22, pages 19–33. USENIX Association, 2022.

[27] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23. Association for Computing Machinery, 2023.

[28] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference*, USENIX ATC '19, pages 947–960. USENIX Association, 2019.

[29] Jinwoo Jeong, Seungsu Baek, and Jeongseob Ahn. Fast and efficient model serving using multi-gpus with direct-host-access. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23. Association for Computing Machinery, 2023.

[30] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, et al. {MegaScale}: Scaling large language model training to more than 10,000 {GPUs}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 745–760, 2024.

[31] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023.

[32] Taeyoon Kim, Suyeon Jeong, Jongseop Lee, Soobee Lee, and Myeongjae Jeon. Sibylla: To retry or not to retry on deep learning job failure. In *2022 USENIX Annual Technical Conference*, USENIX ATC '22, pages 263–270. USENIX Association, 2022.

[33] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23. Association for Computing Machinery, 2023.

[34] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. ModelKeeper: Accelerating DNN training via automated training warmup. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '23, pages 769–785. USENIX Association, 2023.

[35] Van-Hoang Le and Hongyu Zhang. Log parsing with prompt-based few-shot learning. In *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23. IEEE Press, 2023.

[36] Shijian Li, Robert J. Walls, and Tian Guo. Characterizing and Modeling Distributed Training with Transient Cloud GPU Servers. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 943–953, Singapore, Singapore, November 2020. IEEE.

[37] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '23, pages 663–679. USENIX Association, 2023.

[38] Xinyu Lian, Yinfang Chen, Runxiang Cheng, Jie Huang, Parth Thakkar, and Tianyin Xu. Configuration validation with large language models. *CoRR*, abs/2310.09690, 2023.

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[40] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022.

[41] LlamaTeam. The llama 3 herd of models, 2024.

[42] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, Yiqiao Jin, Fan Zhang, Xian Wu, Hanqing Zhao, Dacheng Tao, Philip S. Yu, and Ming Zhang. Large language model agent: A survey on methodology, applications and challenges, 2025.

[43] Zeyang Ma, An Ran Chen, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. Llmparser: An exploratory study on using large language models for log parsing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery.

[44] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '20, pages 289–304. USENIX Association, 2020.

[45] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. Galvatron: Efficient transformer training over multiple gpus using automatic parallelism. *Proceedings of the VLDB Endowment*, 16(3):470–479, November 2022.

[46] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, pages 561–577. USENIX Association, 2018.

[47] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 481–498. USENIX Association, 2020.

[48] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021.

[49] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[50] OpenAI. Gpt-5 system card. 2025.

[51] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 3sigma: Distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of the Thirteenth EuroSys*

Conference, EuroSys '18. Association for Computing Machinery, 2018.

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32 of *NeurIPS '19*. Curran Associates, Inc., 2019.

[53] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, number Article 3 in EuroSys '18, pages 1–14, New York, NY, USA, April 2018. Association for Computing Machinery.

[54] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 1–18. USENIX Association, July 2021.

[55] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, pages 8748–8763, 2021.

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

[57] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[58] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '12. Association for Computing Machinery, 2012.

[59] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[60] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[61] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '18, 2018.

[62] Salvatore Sanfilippo. Redis: An in-memory database. https://redis.io, 2009.

[63] Christoph Schuhmann, Andreas Köpf, Richard Vencu, Theo Coombes, and Romain Beaumont. Laion coco: 600m synthetic captions from laion2b-en, September 2022. Available at: https://laion.ai/blog/laion-coco/.

[64] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference*, USENIX ATC '20, pages 205–218. USENIX Association, 2020.

[65] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, page 1279–1297. ACM, March 2025.

[66] Dharma Shukla, Muthian Sivathanu, Srinidhi Viswanatha, Bhargav Gulavani, Rimma Nehme, Amey Agrawal, Chen Chen, Nipun Kwatra, Ramachandran Ramjee, Pankaj Sharma, Atul Katiyar, Vipul Modi, Vaibhav Sharma, Abhishek Singh, Shreshth Singhal, Kaustubh Welankar, Lu Xun, Ravi Anupindi, Karthik Elangovan, Hasibur Rahman, Zhou Lin, Rahul Seetharaman, Cheng Xu, Eddie Ailijiang, Suresh Krishnappa, and Mark Russinovich. Singularity: Planet-scale, preemptive and elastic scheduling of ai workloads. *CoRR*, abs/2202.07848, 2022.

[67] Amber L Simpson, Michela Antonelli, Spyridon Bakas, Michel Bilello, Keyvan Farahani, Bram Van Ginneken, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, et al. A large annotated medical image dataset for the development and evaluation of segmentation algorithms. *arXiv preprint arXiv:1902.09063*, 2019.

[68] Krishna Srinivasan, Karthik Raman, Jiecao Chen, Michael Bendersky, and Marc Najork. WIT: Wikipedia-based image text dataset for multimodal multilingual machine learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2443–2449. ACM, 2021.

[69] Cheng Tan, Zhichao Li, Jian Zhang, Yu Cao, Sikai Qi, Zherui Liu, Yibo Zhu, and Chuanxiong Guo. Serving dnn models with multi-instance gpus: A case of the reconfigurable machine scheduling problem, 2021.

[70] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *ICML '19*, pages 6105–6114, 2019.

[71] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[72] 5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen,

Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025.
[73] Qwen Team. Introducing qwen1.5, February 2024.
[74] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Lunting Fan, Lingfei Wu, and Qingsong Wen. Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. *CoRR*, abs/2310.16340, 2023.
[75] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '22, pages 945–960. USENIX Association, 2022.
[76] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. Netllm: Adapting large language models for networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, volume 33 of *ACM SIGCOMM '24*, page 661–678. ACM, August 2024.
[77] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, pages 595–610. USENIX Association, 2018.
[78] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 533–548. USENIX Association, 2020.
[79] Lei Xie, Jidong Zhai, Baodong Wu, Yuanbo Wang, Xingcheng Zhang, Peng Sun, and Shengen Yan. Elan: Towards generic and efficient elastic training for deep learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems*, ICDCS '20, pages 78–88. IEEE, 2020.
[80] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024.
[81] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
[82] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. Horus: Interference-aware and prediction-based scheduling in deep learning systems. *IEEE Transactions on Parallel and Distributed Systems*, 33:88–100, 2022.
[83] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer Berlin Heidelberg, 2003.
[84] Shan Yu, Jiarong Xing, Yifan Qiao, Mingyuan Ma, Yangmin Li, Yang Wang, Shuo Yang, Zhiqiang Xie, Shiyi Cao, Ke Bao, Ion Stoica, Harry Xu, and Ying Sheng. Prism: Unleashing gpu sharing for cost-efficient multi-llm serving, 2025.
[85] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models. *CoRR*, abs/2205.01068, 2022.
[86] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, pages 559–578. USENIX Association, 2022.