BIP: ONLINE SCHEDULING FOR LLM BATCH INFERENCE WITH PREDICTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The growing deployment of large language models (LLMs) has created a large demand for offline batch inference, where jobs must be executed under deadlines on heterogeneous GPU clusters. Unlike online serving, this setting is throughput-driven and job-oriented. We introduce BIP, a scheduling framework that combines output-length prediction, rolling-horizon integer programming, and online correction of underestimated workloads. The scheduling model captures job deadlines, GPU-type-dependent speeds, parallel execution, and slot-boundary preemption. We develop three formulations for completed jobs, normalized service, and completion time, and establish the interpretation and feasibility conditions of their objectives. Simulations using batches constructed from instruction–response data show that the best-of-three formulation envelope improves deadline-completion rates over FCFS and SJF across the tested configurations. At normalized load 1.0 and relative job size 1.0, the corrected envelope completes 55.3% more jobs than FCFS and 32.4% more than SJF. These results demonstrate the potential of combining workload predictions with job-level deadline scheduling for offline LLM inference.

1 INTRODUCTION

The rapid adoption of large language models (LLMs) has led to a growing demand for efficient inference across a wide range of applications. Inference workloads typically fall into two categories: online inference (serving) and offline batch inference. Online inference supports interactive, user-facing applications and therefore places strict requirements on service-level objectives (SLOs), particularly time-to-first-token (TTFT) and time-per-output-token (TPOT). When resources are insufficient, online requests may time out or fail, compromising user experience.

In contrast, offline batch inference processes large volumes of user-submitted tasks in a queued, non-interactive manner. A job may contain many independent prompts, for example for document generation, dataset preprocessing, or evaluation. Its value often depends on completing the entire collection by a deadline. The objective is therefore not simply to generate tokens quickly, but to convert a limited GPU budget into as many on-time job completions as possible.

Offline LLM batch inference introduces four scheduling challenges. First, jobs vary substantially in the number and lengths of their constituent prompts. Second, output lengths are unknown before autoregressive decoding, making processing requirements difficult to estimate. Third, jobs arrive asynchronously, so allocations must be revised as new work becomes available. Finally, heterogeneous GPUs provide different processing capacities. Together, these factors require a scheduler to account for predicted workloads, deadlines, and the available resources jointly.

Much recent work on LLM inference scheduling focuses on request-level latency, batching, or speculative execution (Zheng et al., 2023; Qiu et al., 2024; Li et al., 2025; Yu et al., 2022). These systems provide important tools for serving individual requests, but their objectives differ from maximizing the number of complete offline jobs delivered before deadlines. An offline job contributes to our objective only when all of its requests finish by its deadline. Request-level latency minimization therefore does not necessarily maximize job completions, and a shortest-job-first policy can delay a longer job whose deadline is more urgent.

In this work, we investigate *job-oriented scheduling for offline LLM batch inference* on a heterogeneous GPU cluster. Each job has a release time and a completion deadline. Jobs are parallelizable

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

across independent prompts and preemptive at scheduling-slot boundaries. We introduce an integer-programming framework that periodically optimizes GPU allocations over a rolling planning horizon. GPU heterogeneity is represented through type-dependent speed factors, and the integer program is re-solved using the remaining predicted workloads of unfinished jobs. This scheduling abstraction assumes preloaded models and does not include model-loading or request-migration costs.

Effective scheduling requires useful estimates of job processing requirements. Inspired by inference-length prediction (Zheng et al., 2023; Qiu et al., 2024), we fine-tune LLaMA3-8B with LoRA to output a token-count estimate rather than a full response, using discretized response lengths as supervised targets.We then integrate these predictions into the scheduler. When an unfinished job exhausts its predicted workload, an online correction rule increases its effective workload and triggers re-optimization. This feedback updates scheduling inputs rather than retraining the prediction model.

Our contribution is the integration of prediction, deadline-aware job scheduling, and execution feedback within one rolling-horizon framework. We retain three objectives to distinguish full job completion from service allocation and completion-time minimization. Our evaluation compares their configuration-wise best-of-three envelope with FCFS and SJF on synthetic batches constructed from instruction–response data. The contributions are:

- **Problem formulation and scheduling.** We formulate deadline-constrained batch inference on uniform-speed heterogeneous GPUs, with job parallelism and slot-boundary preemption, and develop three integer-programming formulations within a common rolling-horizon framework.
- **Prediction and online correction.** We combine a length predictor with an online correction rule that doubles a job's workload estimate when the job remains unfinished after receiving at least its estimated amount of service. The correction requires no retraining, and we bound the number of such estimate updates per job.
- **Experimental evaluation.** We evaluate output-length prediction and compare deadline-completion rates across nine workload configurations. The results quantify the performance envelope of the formulation family and the effect of enabling correction at the envelope level.

2 Related Work

2.1 Scheduling Methods

Scheduling on heterogeneous parallel machines has been extensively studied in operations research and systems. Classical deadline-constrained scheduling includes computationally difficult variants (Pinedo, 2022). Integer and mixed-integer programming provide a direct way to express machine capacity, processing requirements, and deadline constraints (Floudas and Lin, 2005; Schweissguth et al., 2017).

Scheduling theory distinguishes identical machines (P_m), uniform machines with job-independent speed factors (Q_m), and unrelated machines with job-dependent processing times (R_m) (Pinedo, 2022). Our abstraction is uniform-speed: GPU heterogeneity appears through per-type speeds, while job variability is represented by workloads.

Large-scale systems such as Borg (Verma et al., 2015), Kubernetes (Burns et al., 2016), and Ray (Moritz et al., 2018) address broad cluster-management requirements. Our focus is narrower: explicitly optimizing deadline-constrained job objectives using predicted workloads. The rolling horizon determines how this optimization interacts with online arrivals; it does not by itself provide a guarantee for the full online problem.

2.2 LLM Inference Scheduling

Zheng et al. (2023) developed a sequence-scheduling pipeline that fine-tunes a Vicuna-7B model to predict response lengths, groups instructions with similar predicted lengths, and adjusts batch sizes. Qiu et al. (2024) employed a BERT-based predictor in speculative shortest-job-first (SSJF) scheduling to prioritize requests with shorter estimated outputs. Li et al. (2025) studied semi-clairvoyant

scheduling of speculative-decoding requests, while ORCA (Yu et al., 2022) addresses distributed serving of Transformer-based generative models. These works motivate prediction-aware scheduling. Our scheduling unit is instead a complete offline job, potentially containing many requests, with a job-level deadline. Adapting request-level schedulers to this objective is an important comparison beyond the FCFS and SJF baselines evaluated here.

2.3 Prediction for Inference Length

A common approach to predicting inference length is to use a proxy model to estimate the output length. Zheng et al. (2023) explored ability of LLMs to estimate response lengths and improved this ability through instruction tuning. Qiu et al. (2024) used a BERT-based model with a prediction head. Related length-prediction components also appear in non-autoregressive sequence generation (Lee et al., 2018; 2020; Shu et al., 2020). We build on this line of work by using predicted lengths as workload estimates and correcting them through observed job progress. We do not require the predictor to identify the exact remaining response length after an underestimation event.

3 Problem Statement

We study batch inference jobs on a cluster with GPU types $t \in \mathcal{T} = \{1, \ldots, T\}$. There are m_t GPUs of type t, each providing speed $s_t > 0$ in token-work units per scheduling slot. Jobs arrive in batches at integer times. Jobs in a batch share a release time r_b; we write $r_j = r_b$ for each such job. Job j has deadline d_j and a positive initial workload estimate W_j, obtained by aggregating predicted output lengths over its prompts. Its true workload $W_j^\star$ is unknown to the scheduler. We use d_j to denote the last slot in which processing is permitted, so completion by the end of slot d_j is on time.

The cluster follows a *uniform-speed* model: the predicted processing requirement on one type-t GPU is $p_{j,t} = W_j / s_t$. A job can receive service from multiple GPUs simultaneously, and allocations can change between slots. This models parallel execution of independent prompts, not linear speedup of a single autoregressive sequence. The optimization treats aggregate token-work as divisible within its service model, while GPU counts are integer. The objective is to maximize the number of jobs actually completed by their deadlines, subject to GPU capacity. Positive job weights w_j allow priority-aware variants; setting all $w_j = 1$ gives the completion-count objective.

Uniform-speed assumption and system scope. The workload–speed factorization is the standard Q_m abstraction (Pinedo, 2022); related factorizations appear in asymmetric multicore scheduling (Chen et al., 2012). GPU inference benchmarks motivate representing heterogeneous devices through throughput factors (Chitty-Venkata et al., 2024; Ori, 2024). We use this as a scheduling approximation for a fixed model and inference configuration on an eligible GPU pool. It does not capture job-dependent speed ratios, prompt indivisibility, memory constraints, or nonlinear batching effects. Models are assumed to be preloaded. Model loading, model switching, warmup, active-request migration, and KV-cache movement are not included in the optimization or the scheduling simulations.

4 Scheduling Framework

4.1 Overview

As illustrated in Figure 1, BIP has three modules. **Inference length prediction** estimates each prompt's output token count using an instruction-tuned LLaMA3-8B model. **Data preprocessing and batching** aggregates these counts into job workloads and records release times and deadlines. **ILP-based scheduling** allocates the heterogeneous GPUs over a finite horizon, executes only the first slot, and re-optimizes using the updated set of unfinished jobs. An execution monitor distinguishes actual completion from predicted completion and supplies the feedback used by the correction rule in Section 4.5.

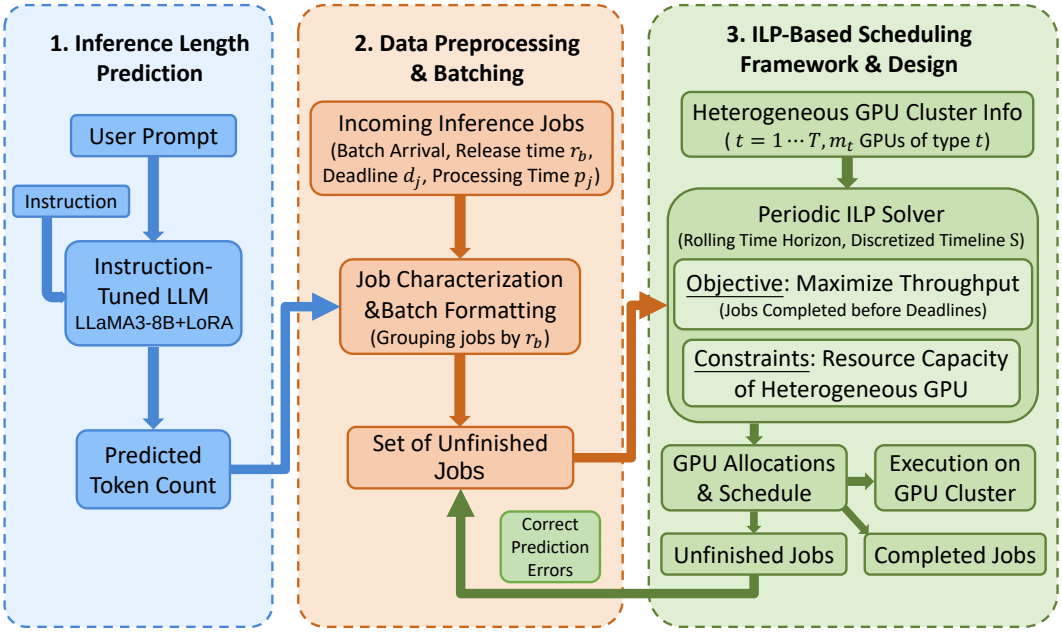

Figure 1: BIP combines length prediction, job characterization, and rolling-horizon scheduling. Execution feedback updates the effective workloads of unfinished jobs; the prediction model is not retrained online. The diagram shows the completion-count objective, with alternative objectives defined in Section 4.3.

4.2 Inference Length Prediction

Accurate inference-length estimation is a key component of our framework. Following prior length-prediction methods (Zheng et al., 2023; Qiu et al., 2024), we use instruction tuning so that the model outputs an estimated token count rather than a complete response.

Data preparation. We sample 10,000 prompts from Alpaca (Taori et al., 2023) for training. For each sampled prompt, we append the length-estimation instruction shown in Table 1. We tokenize the corresponding reference response to obtain its length, quantize that length into 10-token bins, and train the model to output the associated length value rather than the response. Thus the predictor is trained on reference-response lengths; these labels are not measured runtimes or newly generated responses from a deployed inference engine. We aggregate prompt-level predictions to obtain job-level workload estimates.

Training. We use LLaMA3-8B with LoRA adaptation, which introduces low-rank parameter updates while keeping the underlying pretrained weights fixed. This reduces the number of trainable parameters used to adapt the predictor. The prompting format, prediction metrics, and error analysis are provided in Appendix B.

4.3 ILP-Based Scheduling Framework

At epoch $\tau \geq 1$, the scheduler considers the released, unfinished jobs $\mathcal{J}_\tau$ over slots $\mathcal{S}_\tau = \{\tau, \tau + 1, \ldots, \tau + H - 1\}$. Let $\widehat{W}_j^\tau > 0$ be job j's effective workload estimate, initially W_j, and let C_j^τ be its cumulative executed service before slot τ. The remaining predicted work and its normalized fraction are

$$R_j^\tau = \max\{\widehat{W}_j^\tau - C_j^\tau, 0\}, \qquad \mathrm{rem}_j^\tau = R_j^\tau / \widehat{W}_j^\tau. \tag{1}$$

At epoch τ, the effective workloads $\widehat{W}_j^\tau$ and executed service C_j^τ are fixed inputs to the optimization problem over future slots $k \in \mathcal{S}_\tau$. These inputs are updated between scheduling epochs using execution feedback.

GPU allocation and common constraints. The variable $x_{j,t,k} \in \mathbb{Z}_{\geq 0}$ is the number of type-t GPUs assigned to job j in slot k. Each contributes s_t units of service. All three formulations impose

$$\sum_{j \in \mathcal{J}_\tau} x_{j,t,k} \leq m_t \qquad \forall t \in \mathcal{T},\ k \in \mathcal{S}_\tau, \tag{2}$$

$$x_{j,t,k} = 0 \qquad \forall j, t, k \text{ with } k < r_j \text{ or } k > d_j. \tag{3}$$

Write $B_j^\tau(x) = \sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{S}_\tau} s_t x_{j,t,k}$ for the service allocated to j in the new horizon. Only the first-slot allocations are executed. The next solve uses observed progress and completion, new arrivals, and any workload corrections; unexecuted future allocations are discarded.

ILP-1: Maximize weighted completed jobs. For binary indicators z_j, the formulation is

$$\max \sum_{j \in \mathcal{J}_\tau} w_j z_j \quad \text{s.t.} \quad B_j^\tau(x) \geq R_j^\tau z_j \quad (\forall j \in \mathcal{J}_\tau), \qquad z_j \in \{0,1\}. \tag{4}$$

A job contributes only when its predicted remaining requirement is covered by the horizon allocation. Equivalently, dividing the constraint by $\widehat{W}_j^\tau$ gives $B_j^\tau(x)/\widehat{W}_j^\tau \geq \mathrm{rem}_j^\tau z_j$. These indicators certify *predicted* completion, not completion under the unknown true workload. Actual on-time completions are counted by the execution monitor.

ILP-2: Maximize weighted normalized service. The second formulation uses a service-based objective and a workload cap:

$$\max \sum_{j \in \mathcal{J}_\tau} w_j \frac{B_j^\tau(x)}{\widehat{W}_j^\tau} \quad \text{s.t.} \quad B_j^\tau(x) \leq \widehat{W}_j^\tau \quad (\forall j \in \mathcal{J}_\tau). \tag{5}$$

The cap permits at most one effective workload of service per job in the current horizon. This is an *allocated-service surrogate*, not an exact remaining-progress objective: if a job has already received service, the cap is not R_j^τ, and the objective can credit allocations beyond its predicted residual requirement. Integer slot allocations can also leave capacity unused when no whole GPU-slot fits under the cap. ILP-2 has no additional binary indicators, but is still an integer program because $x_{j,t,k}$ is integer.

ILP-3: Minimize weighted completion time under hard deadlines. Using completion-slot variables $c_j \geq 0$ and binary activation variables $y_{j,t,k}$, we solve

$$\min \quad \sum_{j \in \mathcal{J}_\tau} w_j (c_j - d_j), \tag{6}$$

$$\begin{aligned} &\text{s.t.} \quad c_j \geq k\, y_{j,t,k}, && x_{j,t,k} \leq m_t y_{j,t,k} && \forall j, t, k, \\ &\quad B_j^\tau(x) \geq R_j^\tau, && y_{j,t,k} \in \{0,1\} && \forall j, t, k. \end{aligned} \tag{7}$$

The deadlines are fixed, so minimizing (6) is equivalent to minimizing $\sum_j w_j c_j$. Together with (3), this is a completion-time objective with hard deadlines, not a tardiness objective. Every job must receive its remaining predicted work within the horizon. ILP-3 is therefore applicable to jointly feasible active job sets and can be infeasible under overload or a short horizon. Admission control or an infeasibility fallback would be an additional policy component, not part of these constraints.

Variables and operational goals. All formulations share nonnegative integer allocations x and constraints (2)–(3). ILP-1 additionally uses z; ILP-3 uses c and y. Completion slots can be chosen integer at an optimum of ILP-3. The formulations differ in their processing requirements as well as their objectives: ILP-1 selects completions, ILP-2 caps normalized service, and ILP-3 requires all modeled jobs to finish. A complete run uses one fixed formulation.

4.4 Analysis of the ILP Formulations

The common constraints ensure capacity feasibility and prohibit processing outside a job's availability window. They do not ensure that predictions are correct or that every job can meet its deadline. ILP-1 and ILP-2 always admit the zero-allocation solution. ILP-3 does not: with one GPU, one remaining slot, and two jobs each requiring a full slot before the same deadline, its completion requirements are infeasible, while ILP-1 can choose one job.

Proposition 1 (Completion-time interpretation). *Fix an epoch with $R_j^\tau > 0$ and $w_j > 0$ for every modeled job. If ILP-3 is feasible, then at every optimal solution c_j equals the first slot by whose end the cumulative planned service for j reaches R_j^τ.*

The activation constraints force c_j to be at least the last allocated slot. The positive-weight minimization makes this bound tight. If the required work were completed earlier, deleting later allocations would preserve feasibility and strictly improve the objective. Appendix A gives the full proof. The proposition concerns a predicted plan at one epoch, not completion under future arrivals or prediction errors.

ILP-1 directly rewards full predicted completions. ILP-2 rewards service at rate $w_j/\widehat{W}_j^\tau$ and does not imply proportional allocation or a higher completion rate. ILP-3 advances weighted completion times when its all-job requirements are feasible. No robustness ordering among the three follows from these objectives alone. In particular, a common multiplicative workload error can change feasibility relative to fixed deadlines even when job ordering is unchanged.

For $n = |\mathcal{J}_\tau|$, each formulation has nTH allocation variables. ILP-1 adds n binary indicators; ILP-3 adds nTH binary indicators and n completion variables. Rolling-horizon optimization bounds the represented planning window and allows updated predictions, but variable counts do not establish wall-clock solver time or an online approximation guarantee.

4.5 Online Correction of Prediction Errors

Length predictions can overestimate some jobs and severely underestimate others. We therefore maintain the effective workloads $\widehat{W}_j^\tau$ used in (1) and the ILP formulations, without retraining the predictor. Feedback is based on service actually executed and whether the job has finished, rather than on the optimizer's completion indicators.

Early completion. When a job finishes, it is removed from the active set and its future allocations are canceled. At the next scheduling decision, the released capacity is available to waiting jobs. The planning model does not require migration of an active request or within-slot preemption.

Underprediction and geometric backoff. If j remains unfinished after receiving service $C_j^\tau \geq \widehat{W}_j^\tau$, its predicted remaining work is exhausted. We update

$$\widehat{W}_j^\tau \leftarrow 2\widehat{W}_j^\tau \tag{8}$$

and recompute R_j^τ. Repeated underprediction events trigger further doublings. The next rolling-horizon solve uses the corrected effective workloads. Overestimation is handled by observed early completion rather than by reducing estimates before completion.

Number of corrections. Under the token-work service model, suppose an unfinished job has received less than its true required work $W_j^\star$. Starting from $W_j > 0$, the number of doublings is at most

$$N_j \leq \max\left\{0, \left\lceil \log_2 \frac{W_j^\star}{W_j} \right\rceil \right\}. \tag{9}$$

After this many updates, the effective workload is at least the true workload, so no further underprediction event can occur. Appendix A provides the proof. This bound controls the number of estimate updates; it does not guarantee that a job receives enough service or meets its deadline. The rule preserves the linear structure of each optimization problem because workload estimates are updated between solves.

5 Experiments

Our evaluation has two parts: inference-length prediction and job scheduling using predicted lengths. The prediction experiments use instruction–response datasets. The scheduling experiments are simulations of synthetic offline batches constructed from Alpaca responses, not end-to-end measurements on a GPU serving system.

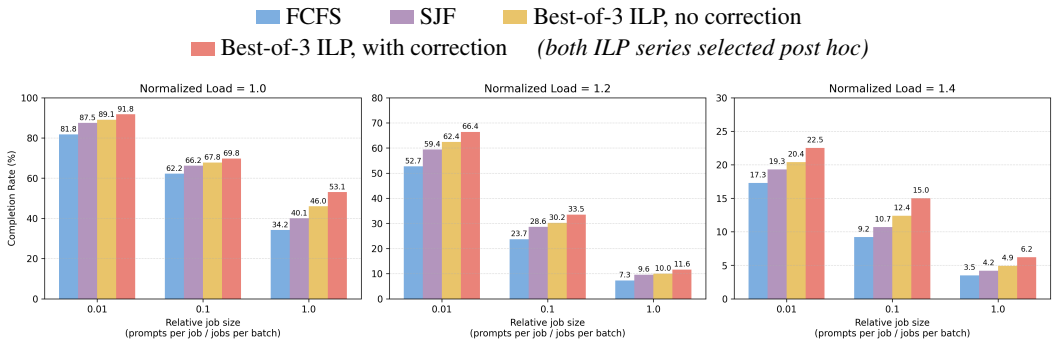

Figure 2: Deadline-completion rates across normalized-load settings and relative job sizes. Each ILP bar is the highest reported completion rate among three fixed-formulation runs for that configuration, selected separately with and without correction. Numerical values are listed in Appendix C.

5.1 Inference Length Prediction

We fine-tune LLaMA3-8B on Alpaca using the prompt and length-bin construction in Section 4.2. The prediction results in Appendix B report Acc-50 and Acc-100, the fractions of predictions within 50 and 100 tokens of the reference length, and mean absolute error (MAE). The Alpaca evaluation shown in Figure 4 contains 1,500 prediction–reference pairs and reports MAE 31.808, Acc-50 81.27%, and Acc-100 95.87%. Table 3 reports the same Alpaca metrics rounded to one decimal place, together with prediction results on No Robots and Dolly 15k. These latter datasets evaluate length prediction only, not downstream scheduling.

5.2 Task Scheduling

Experimental setup and metrics. We construct jobs by sampling prompts and aggregating their predicted and reference-response lengths. Jobs in each batch share release time r_b and have individual deadlines d_j. The simulated cluster contains GPU types with speeds s_t and capacities m_t. Time is discretized into slots. Each ILP run uses predicted workloads to allocate resources; reference-response lengths determine actual completion in the simulator. The completion rate is the fraction of jobs finished by their deadlines.

Figure 2 uses three normalized-load settings and three relative job sizes. The normalized-load index is total predicted batch workload divided by the cluster's aggregate per-slot processing capacity, $\sum_j W_j / \sum_t m_t s_t$; it expresses workload in cluster-slot equivalents rather than a stationary utilization rate. Relative job size is the number of prompts per job divided by the number of jobs per batch. These parameters vary both the work-to-capacity ratio and the concentration of work within jobs.

Baselines. FCFS processes jobs in nondecreasing release-time order and assigns available resources to the earliest waiting jobs. SJF prioritizes shorter workloads and can preempt lower-priority jobs when a shorter job arrives. Both are job-level ordering baselines using an aggregate cluster speed $\sum_t m_t s_t$. This aggregation includes the per-type speeds in our uniform-speed abstraction; it is not evidence that the baselines ignore GPU heterogeneity. The comparison evaluates deadline-aware allocation against these ordering rules, not superiority over complete LLM serving systems.

Formulation-selection protocol. For each workload configuration, ILP-1, ILP-2, and ILP-3 were run separately with identical arrivals, predictions, deadlines, and GPU capacities. Each run kept its objective fixed. We report the highest realized deadline-completion rate among the three, selected after the runs. Selection was performed separately with and without correction. The ILP results therefore form a *configuration-wise, post-hoc best-of-three envelope*, rather than the performance of a single pre-specified online policy. The scheduler itself uses predicted workloads; true lengths determine simulated completion and the subsequent envelope selection. The envelope is not an offline-optimal upper bound, and no held-out objective-selection rule is evaluated.

Results. The corrected and uncorrected ILP envelopes exceed FCFS and SJF in all nine reported configurations. At load 1.0 and relative job size 1.0, the corrected envelope attains 53.1% completion, versus 34.2% for FCFS and 40.1% for SJF. The gains are 18.9 and 13.0 percentage points, respectively, corresponding to relative increases of 55.3% and 32.4%, computed from the displayed rounded values. Completion rates decrease as the load rises and work becomes concentrated in larger jobs.

The corrected envelope is also above the uncorrected envelope in each configuration. At load 1.0 and relative job size 1.0, it increases from 46.0% to 53.1%; at load 1.4 and relative job size 1.0, it increases from 4.9% to 6.2%. These are differences between independently selected envelopes, not paired correction gains for a fixed formulation. The comparison identifies the potential of the formulation family under each condition, but does not establish which objective should be selected before execution or a robustness ranking among the objectives.

6 Limitations and Discussion

BIP uses a uniform-speed, aggregate-work abstraction and assumes preloaded models. Prompt-level execution, memory residency, batching interactions, and loading or migration overheads require additional system modeling. The scheduling results are simulations; they do not measure wall-clock throughput, GPU utilization, or end-to-end serving costs. Prediction performance on reference answers also need not transfer unchanged to responses generated by another model or decoding configuration.

The best-of-three protocol characterizes the formulation family rather than a fixed deployment policy. Fixed-formulation comparisons, a held-out selection protocol, and adaptations of length-prediction schedulers such as those of Zheng et al. (2023) and Qiu et al. (2024) remain important evaluation steps. The reported results do not include solver-runtime scaling or per-formulation feasibility and timeout statistics. In particular, ILP-3 requires a feasible all-job completion plan; an implementation intended for arbitrary loads must specify admission or fallback behavior. These issues are separate from the formulation-correctness and correction-count results established here.

7 Conclusion

We study job-oriented scheduling for offline LLM batch inference with uncertain workloads and deadlines. BIP combines instruction-tuned length prediction, rolling-horizon integer programming, and online workload correction. The three formulations distinguish completed-job throughput, normalized service, and completion time under hard deadlines. Our analysis clarifies their feasibility and objective semantics, while the simulation results show consistent gains of the best-of-three envelopes over FCFS and SJF across the tested configurations. Together, these components connect workload estimation with job-level deadline optimization and adapt resource allocation to observed execution progress.

AI use statement

In this work, we used generative AI tools for manuscript revision and the interpretation and presentation of existing experimental results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

References

Benchmarking llama 3.1 8b instruct on nvidia h100 and a100 chips with the vllm inferencing engine. https://www.ori.co/blog/benchmarking-llama-3.1-8b-instruct-on-nvidia-h100-and-a100-chips-with-the-vllm-inferencing-engine, 2024. Accessed 2025-12-11.

Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57, 2016.

Quan Chen, Yawen Chen, Zhiyi Huang, and Minyi Guo. Wats: Workload-aware task scheduling in asymmetric multi-core architectures. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 249–260, 2012.

Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators, 2024.

Christodoulos A Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139(1):131–162, 2005.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, 2018.

Jason Lee, Raphael Shu, and Kyunghyun Cho. Iterative refinement in the continuous space for non-autoregressive neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1006–1015, 2020.

Ruixiao Li, Fahao Chen, and Peng Li. Semi-clairvoyant scheduling of speculative decoding requests to minimize LLM inference latency. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025*, pages 8554–8562, 2025.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, et al. Ray: A distributed framework for emerging ai applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 561–577, 2018.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Cham, 6 edition, 2022. doi: 10.1007/978-3-031-05921-6.

Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew T. Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Efficient interactive llm serving with proxy model-based sequence length prediction. In *The 5th International Workshop on Cloud Intelligence / AIOps at ASPLOS 2024*, volume 5, pages 1–7, San Diego, CA, USA, 2024. Association for Computing Machinery.

Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla, and Gero Mühl. Ilp-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 8–17, 2017.

Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8846–8853, 2020.

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.

Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, pages 1–17, 2015.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, Carlsbad, CA, July 2022. USENIX Association.

Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. Response length perception and sequence scheduling: an llm-empowered llm inference pipeline. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

A Analysis of the Scheduling Formulations

A.1 Completion-Time Encoding

For a fixed epoch τ, define the cumulative planned service through slot k by

$$P_j(k) = \sum_{t \in \mathcal{T}} \sum_{\ell \in \mathcal{S}_\tau : \ell \le k} s_t x_{j,t,\ell}.$$

When $R_j^\tau > 0$ and the full-horizon completion requirement is feasible, the predicted completion slot is

$$q_j = \min\{k \in \mathcal{S}_\tau : P_j(k) \ge R_j^\tau\}.$$

This definition counts completion at slot boundaries. It does not require service to equal the remaining requirement exactly: the final integer allocation can provide more service than necessary.

Proof of Proposition 1. Fix an optimal feasible solution. Because $R_j^\tau > 0$ and $B_j^\tau(x) \ge R_j^\tau$, job j receives a positive allocation in at least one slot. Let L_j be its last such slot. A positive x_{j,t,L_j} forces $y_{j,t,L_j} = 1$ through $x_{j,t,L_j} \le m_t y_{j,t,L_j}$, and hence $c_j \ge L_j$.

Any activation variable for an allocation equal to zero can be set to zero without violating a constraint. Keeping the allocations fixed, we can therefore choose all remaining activation variables to match positive allocations and set $c_j = L_j$. Since $w_j > 0$, an optimum must already have $c_j = L_j$; otherwise this change strictly decreases the objective.

The full-horizon completion requirement ensures that q_j exists and $q_j \le L_j$. Suppose $q_j < L_j$. Delete every allocation to j after q_j and set the associated activation variables to zero. The allocation through q_j still supplies at least R_j^τ service. Deletion only relaxes capacity constraints and preserves the release-time and deadline constraints. Setting $c_j = q_j$ is then feasible and improves the objective by $w_j(L_j - q_j) > 0$, contradicting optimality. Thus $c_j = L_j = q_j$ for every modeled job. □

The positive-residual assumption matters. A job with $R_j^\tau = 0$ requires no predicted future service, even if it is still unfinished in execution. Such a job is not certified as actually complete by the optimization. With correction enabled, its effective workload is doubled when the underprediction trigger is observed; the proposition applies once its predicted residual is positive. Moreover, the proposition describes an *optimal* solution of a feasible finite-horizon problem. An arbitrary solver incumbent can contain slack completion variables or unnecessary late allocations and is not covered by the statement.

A.2 Feasibility and Objective Distinctions

The zero-allocation solution is feasible for ILP-1 by setting $z_j = 0$, and for ILP-2 directly. In contrast, ILP-3 requires $B_j^\tau(x) \ge R_j^\tau$ for every modeled job. Consider one GPU with speed 1, horizon $\mathcal{S}_\tau = \{\tau\}$, and two jobs with residual workloads 1 and deadline τ. Their combined required service is 2, while capacity is 1. ILP-3 is infeasible, whereas ILP-1 can complete either job. Increasing the horizon need not restore feasibility when the hard deadlines remain unchanged.

The term $-\sum_j w_j d_j$ in (6) is constant. It does not create a penalty for lateness or allow late processing, since (3) forbids allocations after the deadline. In particular, ILP-3 cannot trade a late completion of one job against an on-time completion of another. It orders completion times subject to finishing all modeled jobs within their permitted windows.

For ILP-2, the newly allocated normalized service is $B_j^\tau(x)/\widehat{W}_j^\tau$, while the useful *predicted* remaining progress is at most

$$\frac{\min\{B_j^\tau(x), R_j^\tau\}}{\widehat{W}_j^\tau}.$$

These quantities agree before the residual requirement is exceeded, but can differ after a job has already received service. The workload cap in (5) is therefore a per-horizon service budget, not an exact remaining-work constraint. Also, integer allocations matter: with one speed-1 GPU and a

cap of 0.5 units, no positive whole-slot allocation is permitted by that cap. The formulation should consequently not be interpreted as a convex program or a guarantee of full GPU utilization.

Even a common multiplicative prediction error can affect deadline feasibility. Consider one speed-1 GPU and two jobs released in slot 1, both with deadline 3. If their predicted workloads are 1 and 2, both fit within the three available slots. If their true workloads are 2 and 4, the workload ordering is unchanged, but the second job cannot finish even when given all three slots. Thus preserving relative workload order does not preserve the feasible completion set.

A.3 Bound on Geometric Corrections

Lemma 1. *Suppose job j has true workload $W_j^\star > 0$, initial estimate $W_j > 0$, and its cumulative executed service satisfies $C_j < W_j^\star$ whenever it remains unfinished. If estimates are doubled only when the unfinished job has $C_j \geq \widehat{W}_j$, then the total number of doublings satisfies* (9).

Proof. After q doublings, the estimate is $2^q W_j$. Set

$$q^\star = \max\{0, \lceil \log_2(W_j^\star / W_j) \rceil\}.$$

Then $2^{q^\star} W_j \geq W_j^\star$. Once $q^\star$ updates have occurred, an unfinished job satisfies $C_j < W_j^\star \leq \widehat{W}_j$, so the trigger for another update is false. If a job finishes earlier or receives insufficient further service to trigger another update, the number of corrections is smaller. □

The assumption relates completion monitoring to the token-work service model. In a real serving system, service estimates can themselves be inaccurate; the lemma does not cover an arbitrary mismatch between allocated capacity and actual progress. Nor does it imply a deadline, throughput, or competitive-ratio guarantee.

B Inference Length Prediction

B.1 Data Preparation

We fine-tune LLaMA3-8B using Alpaca (Taori et al., 2023). The response-length distribution in Figure 3 spans from 1 to more than 1,200 tokens and has a long tail. Under a job-ordering scheduler, long jobs can delay short jobs, while aggregation into larger jobs can concentrate workload and increase deadline pressure. The prediction target is the number of tokens in the dataset's reference response, as described in Section 4.2.

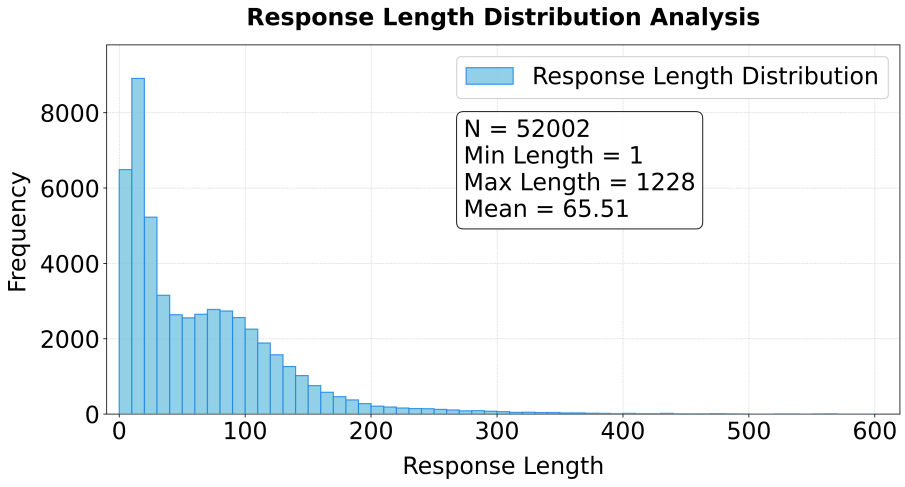

Figure 3: Distribution of response lengths in Alpaca. The displayed horizontal range is truncated at 600 tokens; the dataset includes longer responses.

B.2 Instruction Tuning and Evaluation

Following Zheng et al. (2023), we append a supplementary instruction asking the model to output only a length estimate. Table 1 gives an example. The training set consists of 10,000 sampled Alpaca prompts, with target counts assigned to the nearest size-10 length bin. We evaluate predictions against reference-response token counts.

Prompt:	Give three tips for staying healthy. Do not generate the actual answer to the instruction. Your task is to estimate how many tokens your response would contain and only output that number.
Answer:	50

Table 1: Example of the prompt used for instruction tuning.

For predictions $\widehat{\ell}_i$ and reference lengths ℓ_i, Acc-a is the fraction with $|\widehat{\ell}_i - \ell_i| \leq a$, and MAE is the average absolute error. We report Acc-50, Acc-100, and MAE. The alternative-model prediction results on Alpaca are summarized in Table 2; these models were evaluated in thinking mode. This comparison does not measure their prediction latency or their effect on downstream scheduling.

Model	Acc-50↑	Acc-100↑	MAE↓
Qwen3-32B	75.8%	89.4%	70.8
Qwen3-14B	76.2%	92.3%	51.4
DeepSeek-R1	78.8%	91.0%	43.2

Table 2: Reported length-prediction accuracy of alternative LLMs on Alpaca in thinking mode.

Figure 4 displays 1,500 prediction–reference pairs. The figure reports MAE 31.8080, Acc-50 81.27%, and Acc-100 95.87%; these correspond to the rounded Alpaca row in Table 3. The diagonal denotes perfect prediction. Most points are within the ± 100-token band, while a visible cluster of predictions near 100 tokens shows a tendency to predict similar lengths for different reference responses. Longer reference responses include substantial underestimation errors.

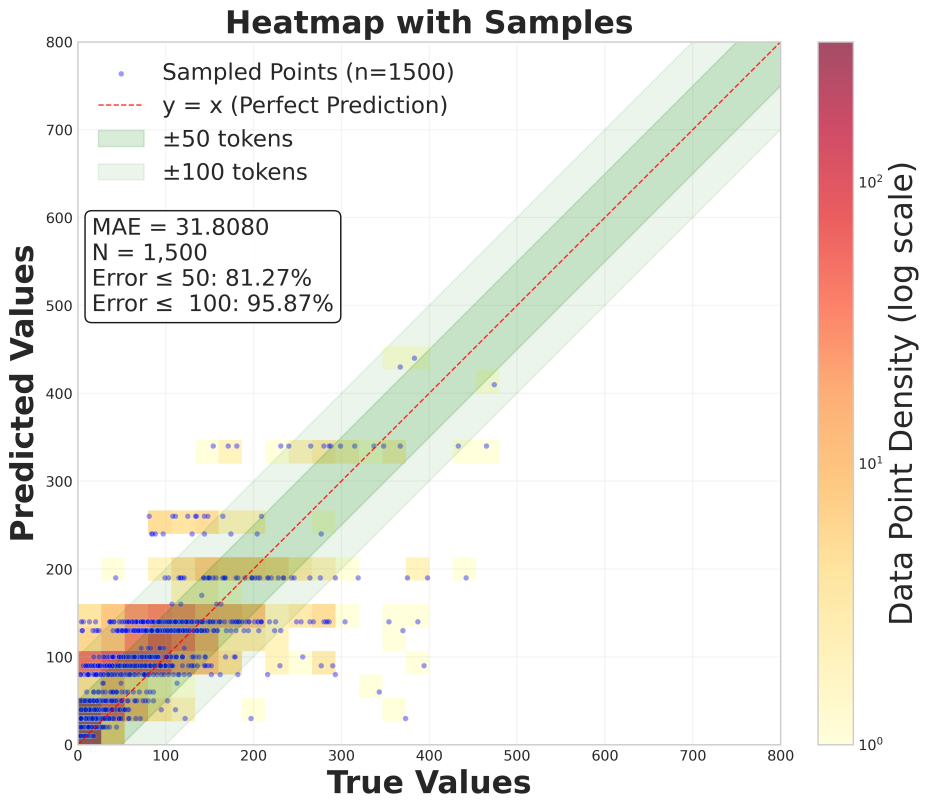

Figure 4: Length predictions against reference-response lengths on Alpaca. The displayed evaluation contains 1,500 pairs; its numerical annotations are reported in the text and rounded in Table 3.

Dataset	Acc-50↑	Acc-100↑	MAE↓
Alpaca	81.3%	95.9%	31.8
No Robots	55.5%	74.4%	78.3
Dolly 15k	69.1%	90.3%	56.8

Table 3: Reported prediction accuracy across datasets. No Robots and Dolly 15k are prediction-only evaluations.

The predictor has lower accuracy on No Robots and Dolly 15k than on Alpaca: MAE increases to 78.3 and 56.8 tokens, respectively. These results show that the quality of workload estimates depends on the dataset. They motivate handling prediction errors in the scheduler, but do not establish downstream scheduling performance on those datasets.

B.3 Error Analysis

Table 4 shows two prompts whose absolute prediction errors exceed 200 tokens. Both are open-ended generation tasks and permit responses with substantially different amounts of detail. In these examples, the predictor underestimates the reference answer's length. Their downstream effect depends on job aggregation, available slack, and scheduling decisions; the examples alone do not quantify that effect. The online correction rule addresses the corresponding scheduling failure mode when an unfinished job exhausts its estimated work.

Prompt:	Generate a recipe for spinach and feta quiche.
Predicted:	120
Reference:	348
Prompt:	Create a jeopardy game based on the subject of computer science.
Predicted:	100
Reference:	387

Table 4: Examples of underpredicted reference-response lengths.

C Numerical Scheduling Results

Table 5 lists the numerical annotations in Figure 2 at their original one-decimal precision. The two ILP columns follow the post-hoc selection protocol in Section 5.2: each is the largest realized completion rate among three fixed-formulation runs, selected independently for the configuration and correction setting. These are the same scheduling results as in the figure, not additional runs or a per-formulation ablation.

Configuration		Deadline-completion rate (%)			
Load	Relative size	FCFS	SJF	Best-of-three envelope	
				No correction	Correction
1.0	0.01	81.8	87.5	89.1	91.8
1.0	0.10	62.2	66.2	67.8	69.8
1.0	1.00	34.2	40.1	46.0	53.1
1.2	0.01	52.7	59.4	62.4	66.4
1.2	0.10	23.7	28.6	30.2	33.5
1.2	1.00	7.3	9.6	10.0	11.6
1.4	0.01	17.3	19.3	20.4	22.5
1.4	0.10	9.2	10.7	12.4	15.0
1.4	1.00	3.5	4.2	4.9	6.2

Table 5: Numerical values plotted in Figure 2. Both rightmost columns are post-hoc formulation envelopes; neither represents a single fixed online policy.