APPROXIMATION ALGORITHMS FOR DUAL-MEMORY REQUEST SCHEDULING IN DISAGGREGATED LLM INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Disaggregated LLM inference executes the prefill and decode phases on separate resource pools, reducing interference between the two phases. However, this architecture introduces a coupled scheduling problem because a request that completes prefill retains its prompt KV cache in prefill memory until it is admitted to the decode pool. Prefill and decode admission decisions therefore jointly determine memory occupancy and request completion times. To the best of our knowledge, this problem lacks a formal formulation, and no approximation guarantees are known for request scheduling in disaggregated LLM inference. We formalize this setting as the dual-memory request scheduling (DMRS) problem, whose objective is to minimize the sum of request completion times, and prove that it is strongly NP-hard. We develop two deterministic polynomial-time approximation algorithms. The first applies to instances satisfying $\alpha < 1$ and $\beta < 1$ and achieves an approximation ratio of $8+\frac{4}{1-\alpha}+\frac{8}{1-\beta}$, where α and β are the maximum normalized prefill and peak decode memory demands among all requests, respectively. In particular, it yields a 32-approximation when $\alpha, \beta \leq 1/2$. The second algorithm applies to all feasible instances and achieves a uniform 36-approximation. Finally, evaluations on two H100 GPUs using trace-derived offline workloads show that the proposed algorithms reduce the sum of request completion times relative to multiple vLLM deployment baselines.

1 INTRODUCTION

Large language model (LLM) inference consists of a prefill phase followed by an autoregressive decode phase. During prefill, the model processes the prompt tokens in parallel and constructs the key-value (KV) cache. During decoding, output tokens are generated sequentially, and the KV cache expands with each token. Prefill is generally compute-bound, whereas decoding is more sensitive to memory bandwidth and KV cache capacity (Patel et al., 2024; Zhong et al., 2024). When colocated on the same GPUs, the two phases compete for shared resources. To reduce this interference, recent systems execute the prefill and decode phases on separate GPU instances or resource pools (Patel et al., 2024; Zhong et al., 2024; Hu et al., 2024b; Qin et al., 2026; Jiang et al., 2025). This architecture allows resources to be allocated independently to the two phases.

However, prefill-decode disaggregation does not eliminate the scheduling dependence between the two phases. After completing prefill, a request may be unable to enter decoding immediately if the available decode memory is insufficient. While waiting, the request retains its prompt KV cache in prefill memory, prolonging its occupancy of that memory and potentially blocking subsequent prefill admissions. Upon decode admission, the KV cache is transferred to decode memory, and the retained prefill memory is released. The request's decode memory usage then grows as output tokens are generated. Consequently, prefill and decode admission decisions are coupled through their effects on memory occupancy in the two resource pools.

Existing systems work has investigated serving architectures and resource allocation (Zhong et al., 2024; Patel et al., 2024; Hu et al., 2024b; Jiang et al., 2025), request scheduling and load balancing to improve goodput and meet service-level objectives (Wang et al., 2025a; Wu et al., 2025; Jin et al., 2024), and KV cache transfer, storage, and management (Qin et al., 2026; Hu et al., 2024a; Luo

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.

et al., 2026; Hong et al., 2025). These studies focus primarily on the design and implementation of practical serving systems. Related theoretical work has analyzed latency-oriented request scheduling subject to a unified memory capacity constraint (Jaillet et al., 2025; Ao et al., 2025; Wang et al., 2025b; Feng et al., 2026b).

To the best of our knowledge, no prior theoretical scheduling model jointly considers prefill and decode admission decisions under separate memory capacities, and no approximation guarantees are known for minimizing the sum of request completion times in this setting. We address this gap by studying offline scheduling of a batch of requests under prefill-decode disaggregation. Our main contributions are summarized as follows.

Contribution 1: We develop a system model for prefill and decode admission scheduling in disaggregated LLM inference. Based on this model, we formulate the dual-memory request scheduling (DMRS) problem with the objective of minimizing the sum of request completion times. We prove that DMRS is strongly NP-hard.

Contribution 2: We develop a deterministic polynomial-time approximation algorithm for instances satisfying $\alpha < 1$ and $\beta < 1$. It achieves an approximation ratio of $8 + \frac{4}{1-\alpha} + \frac{8}{1-\beta}$, where α and β are the maximum normalized prefill and peak decode memory demands, respectively. In particular, it yields a 32-approximation when $\alpha, \beta \leq 1/2$.

Contribution 3: We develop a second deterministic polynomial-time algorithm that applies to every feasible instance, including $\alpha = 1$ or $\beta = 1$, and achieves a uniform 36-approximation.

Contribution 4: We evaluate both algorithms on synthetic instances and workloads derived from the Azure LLM Inference Trace 2023 (Patel et al., 2024). The CPU experiments compare the algorithms' objective values with the exact optimum obtained from a time-indexed integer linear programming (ILP) formulation and with a linear programming (LP) lower bound. Serving experiments on two H100 GPUs use Qwen2.5 (7B) (Yang et al., 2024) and Llama 3.1 (8B) (Grattafiori et al., 2024) to compare total completion time against three vLLM baselines.

2 System Model and Formulation

In this section, we first present the system model. We then formulate DMRS, establish its computational complexity, and derive an LP relaxation for our approximation analysis. Appendix A summarizes the notation used throughout the paper.

2.1 System Model

In our model, the prefill and decode phases execute on separate resource pools with respective memory capacities $M_P, M_D \in \mathbb{Z}_{>0}$.

Execution Setup. Time is divided into slots $[t, t+1)$, where $t \in \{0, 1, 2, \ldots\}$. KV cache memory is measured in token units, with one unit representing the memory required to store the key and value states of one token. During prefill, each active request processes its prompt for a fixed number of slots. During decoding, each active request completes one decode iteration and generates one output token per slot. Multiple requests may execute concurrently in each phase, subject to the corresponding memory capacity. Execution is non-preemptive within each phase: once admitted to a phase, a request remains active until it completes that phase.

Inference Requests. Let $\mathcal{I} = \{1, \ldots, n\}$ denote the set of requests. Each request $i \in \mathcal{I}$ is characterized by $(m_i, p_i, d_i) \in \mathbb{Z}^3_{>0}$, where m_i is its effective prefill processing time, measured in advance under the given deployment configuration and expressed in slots; p_i is the number of input tokens in its prompt and, equivalently, its prefill memory demand in token units; and d_i is the number of output tokens to be generated and, equivalently, the number of decode iterations. After generating u output tokens, request i has decode memory demand $r_i(u) = p_i + u$. Its peak decode memory demand is $h_i = r_i(d_i) = p_i + d_i$.

Throughout the paper, we assume that $p_i \leq M_P$ and $h_i \leq M_D$ for every $i \in \mathcal{I}$. These conditions are necessary and sufficient for feasibility, since the requests can always be processed sequentially. Let $\alpha := \max_{i \in \mathcal{I}} p_i / M_P$ and $\beta := \max_{i \in \mathcal{I}} h_i / M_D$ denote the maximum normalized prefill and peak decode memory demands, respectively. The feasibility conditions imply $\alpha, \beta \leq 1$.

System Overview. As shown in Figure 1, we consider a batch of requests released at time $t = 0$, with their characteristics (m_i, p_i, d_i) known to the scheduler in advance. During execution, each unfinished request belongs to one of four state sets.

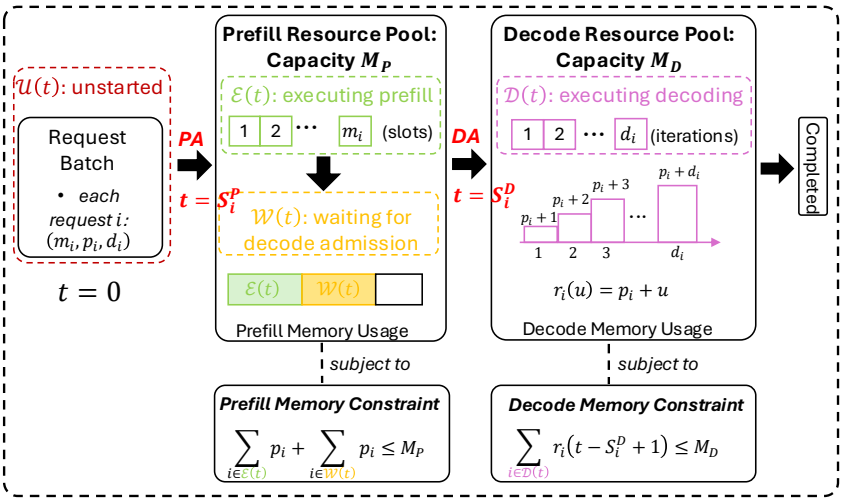


Figure 1: System model for joint prefill and decode admission scheduling.

For each slot $[t, t+1)$, let $\mathcal{U}(t)$, $\mathcal{E}(t)$, $\mathcal{W}(t)$, and $\mathcal{D}(t)$ denote, respectively, the sets of unstarted requests awaiting prefill admission (PA), requests executing prefill, requests awaiting decode admission (DA), and requests executing decoding. Let S_i^P and S_i^D denote the prefill and decode admission times of request i, respectively.

PA and Prefill Memory Constraint. The scheduler selects a feasible subset of unstarted requests for prefill admission. A request i admitted at time S_i^P enters the prefill pool and belongs to $\mathcal{E}(t)$ while executing prefill. It requires m_i slots to process prompt tokens in parallel and generate its prompt KV cache, and it occupies p_i units of prefill memory. Requests not admitted remain in $\mathcal{U}(t)$.

After completing prefill, request i is either admitted immediately to decoding or enters $\mathcal{W}(t)$ while awaiting decode admission. During this waiting period, it retains its prompt KV cache and continues to occupy p_i units of prefill memory. Therefore, request i occupies p_i units of prefill memory throughout $[S_i^P, S_i^D)$. The prefill memory constraint during slot $[t, t+1)$ is thus

$$\sum_{i\in\mathcal{E}(t)} p_i + \sum_{i\in\mathcal{W}(t)} p_i \le M_P.$$

DA and Decode Memory Constraint. At admission time S_i^D, request i enters the decode pool. Its prompt KV cache is transferred from prefill memory to decode memory, and its prefill memory is released. We model this transfer as instantaneous and omit transfer time and network bandwidth constraints. During decoding, request i belongs to $\mathcal{D}(t)$ and completes one iteration per slot.

After generating u output tokens, request i has decode memory demand $r_i(u) = p_i + u$. For $u = 0, \ldots, d_i$, this demand grows from p_i at decode admission to the peak value $h_i = r_i(d_i) = p_i + d_i$. If request i executes decoding during slot $[t, t+1)$, it has generated $t - S_i^D$ output tokens before the slot and generates one additional token during the slot. Its decode memory demand therefore reaches $r_i(t - S_i^D + 1)$ during that slot. Hence, the decode memory constraint is

$$\sum_{i\in\mathcal{D}(t)} (p_i + t - S_i^D + 1) \le M_D.$$

After its d_i-th decode iteration, request i leaves the system and releases its decode memory.

2.2 DMRS Formulation

Based on the system model above, we formulate joint prefill and decode admission scheduling as the dual-memory request scheduling (DMRS) problem, whose objective is to minimize the sum of

request completion times. The optimization problem, denoted by $(\mathbf{O})$, is formulated as follows:

$$\min \quad \sum_{i\in\mathcal{I}} C_i \tag{1a}$$

$$\text{s.t.} \quad C_i^P = S_i^P + m_i, \quad \forall i \in \mathcal{I}, \tag{1b}$$

$$C_i = S_i^D + d_i, \quad \forall i \in \mathcal{I}, \tag{1c}$$

$$S_i^D \geq C_i^P, \quad \forall i \in \mathcal{I}, \tag{1d}$$

$$\sum_{i\in\mathcal{I}} p_i \, \mathbf{1}\{S_i^P \leq t < S_i^D\} \leq M_P, \quad \forall t \in \{0,1,2,\ldots\}, \tag{1e}$$

$$\sum_{i\in\mathcal{I}} \left(p_i + t - S_i^D + 1\right) \mathbf{1}\{S_i^D \leq t < C_i\} \leq M_D, \quad \forall t \in \{0,1,2,\ldots\}, \tag{1f}$$

$$S_i^P, S_i^D \in \mathbb{Z}_{\geq 0}, \quad \forall i \in \mathcal{I}. \tag{1g}$$

For each request $i \in \mathcal{I}$, the scheduler determines its prefill and decode admission times S_i^P and S_i^D. Constraints (1b) and (1c) define its prefill completion time and request completion time, respectively. Constraint (1d) ensures that request i is admitted to decoding after completing prefill. Constraints (1e) and (1f) enforce the prefill and decode memory capacities, respectively, where $\mathbf{1}\{\cdot\}$ denotes the indicator function. Constraint (1g) restricts admission times to nonnegative integers.

Since all requests are released at time zero, C_i is also the end-to-end latency of request i, i.e., minimizing $\sum_{i\in\mathcal{I}} C_i$ is equivalent to minimizing the average end-to-end latency. Let OPT denote the optimal objective value of $(\mathbf{O})$.

Theorem 1. *DMRS is strongly NP-hard, even when every request requires exactly one slot of prefill processing and one decode iteration. (The complete proof is given in Appendix B.)*

2.3 LP Relaxation

For the approximation analysis, we derive an LP relaxation from necessary conditions satisfied by every feasible schedule. The relaxation captures the mandatory memory-time loads imposed by each request on the prefill and decode memory pools. Its optimal value provides a lower bound on OPT.

Prefill Memory-Time Load. During its m_i slots of prefill processing, request i occupies p_i units of prefill memory. Its mandatory prefill memory-time load is $\ell_i^P := p_i m_i$.

Decode Memory-Time Load. During its u-th decode iteration, request i requires $p_i + u$ units of decode memory. Its mandatory decode memory-time load is $\ell_i^D := \sum_{u=1}^{d_i}(p_i+u) = p_i d_i + \frac{d_i(d_i+1)}{2}$.

Let $\bar{C}_i$ denote the relaxed completion-time variable associated with request i. We consider the following linear program, denoted by $(\mathbf{LP})$:

$$\min \quad \sum_{i\in\mathcal{I}} \bar{C}_i \tag{2a}$$

$$\text{s.t.} \quad \bar{C}_i \geq m_i + d_i, \qquad \forall i \in \mathcal{I}, \tag{2b}$$

$$\sum_{i\in\mathcal{S}} \ell_i^P \bar{C}_i \geq \frac{1}{2M_P}\left[\left(\sum_{i\in\mathcal{S}} \ell_i^P\right)^2 + \sum_{i\in\mathcal{S}} (\ell_i^P)^2\right], \quad \forall\, \emptyset \neq \mathcal{S} \subseteq \mathcal{I}, \tag{2c}$$

$$\sum_{i\in\mathcal{S}} \ell_i^D \bar{C}_i \geq \frac{1}{2M_D}\left[\left(\sum_{i\in\mathcal{S}} \ell_i^D\right)^2 + \sum_{i\in\mathcal{S}} (\ell_i^D)^2\right], \quad \forall\, \emptyset \neq \mathcal{S} \subseteq \mathcal{I}. \tag{2d}$$

Let $\mathrm{LP}^\star$ denote the optimal value of $(\mathbf{LP})$. Constraint (2b) provides a lower bound on each request's completion time. Constraints (2c) and (2d) provide lower bounds derived from the mandatory prefill and decode memory-time loads of every nonempty subset of requests, respectively. Lemma 1 establishes that these constraints are satisfied by the completion times of every feasible schedule.

Lemma 1. *For every feasible schedule of $(\mathbf{O})$, setting $\bar{C}_i = C_i$ for all $i \in \mathcal{I}$ yields a feasible solution to $(\mathbf{LP})$. Consequently, $\mathrm{LP}^\star \leq \mathrm{OPT}$. (The complete proof is given in Appendix C.1.)*

Lemma 2. *The linear program* (**LP**) *can be solved in polynomial time. (The complete proof is given in Appendix C.2.)*

3 Approximation for Restricted Instances

Algorithm 1 applies to instances satisfying $\alpha < 1$ and $\beta < 1$. Hence, every request satisfies $M_P - p_i \geq (1-\alpha)M_P$ and $M_D - h_i \geq (1-\beta)M_D$. If a request is rejected by the prefill admission rule, the prefill memory occupied by requests exceeds $(1-\alpha)M_P$. Similarly, if a request is rejected by the decode admission rule, the peak decode memory reserved by requests exceeds $(1-\beta)M_D$.

3.1 Algorithm Design

Using an optimal LP solution $\{\bar{C}_i\}_{i\in\mathcal{I}}$, Algorithm 1 partitions the requests into dyadic groups $\mathcal{G}_q := \{i \in \mathcal{I} : 2^{q-1} < \bar{C}_i \leq 2^q\}$ for $q \in \mathbb{Z}$.

Algorithm 1 LP-Guided Dyadic Group Scheduling

Require: Requests $\{(m_i, p_i, d_i)\}_{i\in\mathcal{I}}$ and memory capacities M_P and M_D
Ensure: Admission times $\{(S_i^P, S_i^D)\}_{i\in\mathcal{I}}$

Solve (**LP**) and obtain an optimal solution $\{\bar{C}_i\}_{i\in\mathcal{I}}$; Construct all nonempty dyadic groups $\mathcal{G}_q$
$t \leftarrow 0$; $\mathcal{E}, \mathcal{W}, \mathcal{D} \leftarrow \emptyset$
for each nonempty $\mathcal{G}_q$, in increasing order of q **do**
  $\mathcal{U}_q \leftarrow \mathcal{G}_q$
  while $\mathcal{U}_q \cup \mathcal{E} \cup \mathcal{W} \cup \mathcal{D} \neq \emptyset$ **do**
    Remove every request $i \in \mathcal{D}$ satisfying $S_i^D + d_i = t$
    Move every request $i \in \mathcal{E}$ satisfying $S_i^P + m_i = t$ to $\mathcal{W}$
    $\widehat{\mathcal{W}} \leftarrow \mathcal{W}$
    for each $i \in \widehat{\mathcal{W}}$, in nondecreasing order of $m_i + d_i$ **do**
      if $\sum_{j\in\mathcal{D}} h_j + h_i \leq M_D$ **then** ▷ reserve peak decode demands
        $S_i^D \leftarrow t$; $\mathcal{W} \leftarrow \mathcal{W} \setminus \{i\}$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{i\}$
      end if
    end for
    $\widehat{\mathcal{U}}_q \leftarrow \mathcal{U}_q$
    for each $i \in \widehat{\mathcal{U}}_q$, in nondecreasing order of $m_i + d_i$ **do**
      if $\sum_{j\in\mathcal{E}} p_j + \sum_{j\in\mathcal{W}} p_j + p_i \leq M_P$ **then**
        $S_i^P \leftarrow t$; $\mathcal{U}_q \leftarrow \mathcal{U}_q \setminus \{i\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{i\}$
      end if
    end for
    if $\mathcal{U}_q \cup \mathcal{E} \cup \mathcal{W} \cup \mathcal{D} \neq \emptyset$ **then**
      $t \leftarrow \min\left(\{S_i^P + m_i : i \in \mathcal{E}\} \cup \{S_i^D + d_i : i \in \mathcal{D}\}\right)$
    end if
  end while
end for

The nonempty groups are processed in increasing order of q. No request from a later group is admitted before all requests in the current group have completed. In each group, we order the requests by nondecreasing $m_i + d_i$, with ties broken by request index.

At each event time, Algorithm 1 performs decode admission before prefill admission. It reserves h_i for each request executing decoding until completion, while its actual decode memory demand satisfies $r_i(u) \leq h_i$ for $0 \leq u \leq d_i$. Appendix D.1 provides a line-by-line description of Algorithm 1.

3.2 Approximation Guarantee

The analysis partitions the slots before each dyadic prefix completes into four cases: a request remains waiting for decode admission, a request remains unadmitted to prefill, or the current group is in its final prefill or decode tail. Failed admissions yield lower bounds on reserved decode memory

or occupied prefill memory, allowing the LP prefix load bounds to control the first two cases; the reserved decode memory-time load is less than twice the mandatory decode load. The dyadic group bounds control the two tails. The lemmas and charging argument appear in Appendix E.

Theorem 2. *For every instance satisfying $\alpha < 1$ and $\beta < 1$, Algorithm 1 is a deterministic polynomial-time $\left(8 + \frac{4}{1-\alpha} + \frac{8}{1-\beta}\right)$-approximation.*

Proof. For each $k \in \mathbb{Z}$, define the dyadic prefix $\mathcal{G}_{\le k} := \bigcup_{q \le k} \mathcal{G}_q$. For k such that $\mathcal{G}_{\le k} \neq \emptyset$, let T_k denote the time at which requests in $\mathcal{G}_{\le k}$ have completed. Lemma 3 bounds the mandatory prefill and decode memory-time loads of this prefix by $2M_P 2^k$ and $2M_D 2^k$, respectively. Lemma 4 shows that the reserved decode memory-time load is less than twice the mandatory decode load. Using these bounds, the charging argument in Lemma 5 shows that requests in $\mathcal{G}_{\le k}$ complete by time

$$T_k \le \left(4 + \frac{2}{1-\alpha} + \frac{4}{1-\beta}\right) 2^k.$$

For any request $i \in \mathcal{G}_k$, the group definition gives $2^{k-1} < \bar{C}_i$, and hence $2^k < 2\bar{C}_i$. Since $C_i \le T_k$,

$$C_i \le \left(8 + \frac{4}{1-\alpha} + \frac{8}{1-\beta}\right) \bar{C}_i.$$

Summing over all requests and applying Lemma 1, which gives $\sum_i \bar{C}_i = \mathrm{LP}^\star \le \mathrm{OPT}$, proves the stated approximation ratio.

By Lemma 2, $(\mathbf{LP})$ can be solved in polynomial time. The grouping, sorting, and event-driven scheduling are also polynomial. Hence, Algorithm 1 is deterministic and runs in polynomial time. □

Corollary 1. *If $\alpha, \beta \le 1/2$, equivalently, if $p_i \le M_P/2$ and $h_i \le M_D/2$ for every $i \in \mathcal{I}$, then Algorithm 1 is a 32-approximation. More generally, the approximation bound tends to 20 as $\alpha, \beta \to 0$ and diverges as either parameter approaches 1 from below.*

4 Approximation for General Instances

To obtain a uniform guarantee, we develop a 36-approximation for every feasible instance, including the boundary cases $\alpha = 1$ or $\beta = 1$. Algorithm 2 combines a classification based on prefill and peak decode memory demands with geometric grouping based on an optimal LP solution.

4.1 Request Classification

We classify each request by its prefill and peak decode memory demands:

$$\mathcal{L}_P := \left\{ i \in \mathcal{I} : p_i > \frac{M_P}{2} \right\}, \quad \mathcal{S}_P := \left\{ i \in \mathcal{I} : p_i \le \frac{M_P}{2} \right\},$$
$$\mathcal{L}_D := \left\{ i \in \mathcal{I} : h_i > \frac{M_D}{2} \right\}, \quad \mathcal{S}_D := \left\{ i \in \mathcal{I} : h_i \le \frac{M_D}{2} \right\}.$$

Requests in $\mathcal{L}_P$ and $\mathcal{S}_P$ are called *prefill-large* and *prefill-small*, while those in $\mathcal{L}_D$ and $\mathcal{S}_D$ are called *decode-large* and *decode-small*, respectively.

The sets $\mathcal{C}^{LL} := \mathcal{L}_P \cap \mathcal{L}_D$, $\mathcal{C}^{SL} := \mathcal{S}_P \cap \mathcal{L}_D$, $\mathcal{C}^{LS} := \mathcal{L}_P \cap \mathcal{S}_D$, and $\mathcal{C}^{SS} := \mathcal{S}_P \cap \mathcal{S}_D$ partition $\mathcal{I}$. In each superscript, the first and second letters indicate the prefill and decode classifications, respectively. The strict $1/2$ threshold ensures that at most one prefill-large request belongs to $\mathcal{E}(t) \cup \mathcal{W}(t)$ and, under peak decode reservation, at most one decode-large request belongs to $\mathcal{D}(t)$.

4.2 Algorithm Design

Fix a geometric grouping ratio $\rho > 1$, let $B_q = \rho^q$, and define $\mathcal{H}_q := \{i \in \mathcal{I} : B_q/\rho < \bar{C}_i \le B_q\}$ for $q \in \mathbb{Z}$.

Algorithm 2 processes the nonempty groups in increasing order of q, completing each group before starting the next. Within each group $\mathcal{H}_q$, it processes $\mathcal{H}_q \cap \mathcal{C}^{LL}$, $\mathcal{H}_q \cap \mathcal{C}^{SL}$, $\mathcal{H}_q \cap \mathcal{C}^{LS}$, and $\mathcal{H}_q \cap \mathcal{C}^{SS}$ in this order. The first subset is processed sequentially, while the other three use event-driven admission subject to both memory capacities. Within each subset, requests are ordered by nondecreasing $m_i + d_i$, with ties broken by request index. For decode admission, the algorithm reserves h_j for every request $j \in \mathcal{D}$ until that request completes. Thus, decode admission requires $\sum_{j \in \mathcal{D}} h_j + h_i \le M_D$ and is not based on the current decode memory usage. Appendix D.2 provides a detailed line-by-line description of Algorithm 2.

Algorithm 2 LP-Guided Geometric Group Scheduling

Require: Requests $\{(m_i, p_i, d_i)\}_{i \in \mathcal{I}}$, memory capacities M_P, M_D, and grouping ratio $\rho > 1$
Ensure: Admission times $\{(S_i^P, S_i^D)\}_{i \in \mathcal{I}}$

Solve $(\mathbf{LP})$ and obtain an optimal solution $\{\bar{C}_i\}_{i \in \mathcal{I}}$; Construct the classes $\mathcal{C}^{LL}, \mathcal{C}^{SL}, \mathcal{C}^{LS}, \mathcal{C}^{SS}$
Construct all nonempty geometric groups $\mathcal{H}_q$
$t \leftarrow 0$; $\mathcal{E}, \mathcal{W}, \mathcal{D} \leftarrow \emptyset$
for each nonempty $\mathcal{H}_q$, in increasing order of q **do**
  for each $i \in \mathcal{H}_q \cap \mathcal{C}^{LL}$, in nondecreasing order of $m_i + d_i$ **do**
    $S_i^P \leftarrow t$; $S_i^D \leftarrow t + m_i$; $t \leftarrow t + m_i + d_i$
  end for
  for each $\mathcal{C}$ in $(\mathcal{C}^{SL}, \mathcal{C}^{LS}, \mathcal{C}^{SS})$ **do**
    $\mathcal{U}_{q,\mathcal{C}} \leftarrow \mathcal{H}_q \cap \mathcal{C}$
    while $\mathcal{U}_{q,\mathcal{C}} \cup \mathcal{E} \cup \mathcal{W} \cup \mathcal{D} \neq \emptyset$ **do**
      Remove every $i \in \mathcal{D}$ satisfying $S_i^D + d_i = t$
      Move every $i \in \mathcal{E}$ satisfying $S_i^P + m_i = t$ to $\mathcal{W}$
      $\widehat{\mathcal{W}} \leftarrow \mathcal{W}$
      for each $i \in \widehat{\mathcal{W}}$, in nondecreasing order of $m_i + d_i$ **do**
        if $\sum_{j \in \mathcal{D}} h_j + h_i \le M_D$ **then** ▷ reserve peak decode demands
          $S_i^D \leftarrow t$; $\mathcal{W} \leftarrow \mathcal{W} \setminus \{i\}$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{i\}$
        end if
      end for
      $\widehat{\mathcal{U}}_{q,\mathcal{C}} \leftarrow \mathcal{U}_{q,\mathcal{C}}$
      for each $i \in \widehat{\mathcal{U}}_{q,\mathcal{C}}$, in nondecreasing order of $m_i + d_i$ **do**
        if $\sum_{j \in \mathcal{E}} p_j + \sum_{j \in \mathcal{W}} p_j + p_i \le M_P$ **then**
          $S_i^P \leftarrow t$; $\mathcal{U}_{q,\mathcal{C}} \leftarrow \mathcal{U}_{q,\mathcal{C}} \setminus \{i\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{i\}$
        end if
      end for
      if $\mathcal{U}_{q,\mathcal{C}} \cup \mathcal{E} \cup \mathcal{W} \cup \mathcal{D} \neq \emptyset$ **then**
        $t \leftarrow \min\big(\{S_i^P + m_i : i \in \mathcal{E}\} \cup \{S_i^D + d_i : i \in \mathcal{D}\}\big)$
      end if
    end while
  end for
end for

4.3 Approximation Guarantee

The proof assigns each execution slot to one of three types. The slot either processes a request that is large in the relevant memory pool, occurs while a small request is blocked by the corresponding admission constraint, or belongs to the final prefill or decode tail of a class phase. The LP bounds on the cumulative loads of each geometric prefix, together with the peak reservation bound, control the first two types. Geometric grouping bounds the accumulated tails.

Appendix F gives the complete slot classification and charging argument.

Theorem 3. *With $\rho = 3/2$, Algorithm 2 is a deterministic polynomial-time 36-approximation for every feasible DMRS instance.*

Proof. For any nonempty group $\mathcal{H}_k$, define $\mathcal{H}_{\le k} := \bigcup_{q \le k} \mathcal{H}_q$ and let $T_k := \max_{i \in \mathcal{H}_{\le k}} C_i$. Lemma 6 bounds the mandatory prefill and decode memory-time loads of this prefix by $2M_P B_k$ and $2M_D B_k$, respectively. Using these bounds, the charging argument in Lemma 7 gives $T_k \le \left(12 + \frac{4\rho}{\rho-1}\right) B_k$.

For every $i \in \mathcal{H}_k$, the group definition gives $B_k < \rho \bar{C}_i$, while $C_i \le T_k$. Therefore, $C_i < \left(12\rho + \frac{4\rho^2}{\rho-1}\right) \bar{C}_i$. Let $f(\rho) := 12\rho + \frac{4\rho^2}{\rho-1}$. For every $\rho > 1$, $f(\rho) - 36 = \frac{4(2\rho-3)^2}{\rho-1} \ge 0$, with equality at $\rho = 3/2$. Hence, by Lemma 1, $\sum_{i \in \mathcal{I}} C_i < 36 \sum_{i \in \mathcal{I}} \bar{C}_i = 36\,\mathrm{LP}^\star \le 36\,\mathrm{OPT}$.

Lemma 2, together with the deterministic polynomial-time grouping and scheduling steps, implies that Algorithm 2 is deterministic and polynomial-time. □

5 Experiments

CPU Evaluation. We compare the objective values of Algorithms 1 and 2 with the exact optimum computed by the ILP in Appendix G and the LP lower bound $\mathrm{LP}^\star$ defined in Section 2.3.

GPU Evaluation. We compare the total completion time (TCT) of Algorithms 1 and 2 with three vLLM baselines: disaggregated prefill and decode serving (P/D), colocated serving (Colocated), and colocated serving with chunked prefill (Colocated Chunked).

5.1 CPU Evaluation

Setup. We implement Algorithms 1 and 2 in Python 3.12.12 and solve the ILP and LP using Gurobi 13.0.1 on an ARM64 macOS machine. We consider synthetic instances and conversation and code workloads derived from the Azure LLM Inference Trace 2023 (Patel et al., 2024).

Table 1: ALG/OPT and ALG/LP for $\kappa = 2$, where $M_P = \lceil \kappa \max_i p_i \rceil$ and $M_D = \lceil \kappa \max_i h_i \rceil$.

Workload	Scale	Algorithm 1	Algorithm 2
Synthetic, ALG/OPT	$n = 10$	**1.660**	**1.945**
Synthetic, ALG/LP	$n = 250$	**1.749**	**1.938**
Conversation, ALG/LP	$n = 250$	**1.491**	**1.685**
Code, ALG/LP	$n = 250$	**1.695**	**2.109**

Results. Algorithm 1 has a lower ratio than Algorithm 2 in each reported setting. All ratios are below the worst-case guarantees. Further scaling and capacity results appear in Appendix H.

5.2 GPU Evaluation

Setup. We conduct serving experiments on two NVIDIA H100 GPUs using Qwen2.5 (7B) (Yang et al., 2024) and Llama 3.1 (8B) (Grattafiori et al., 2024). We use separate offline batches of conversation and code requests derived from the Azure LLM Inference Trace 2023 (Patel et al., 2024).

Table 2: TCT normalized by the corresponding P/D result for $n = 250$ and $\kappa = 1.5$.

Model	Workload	P/D	Colocated	Colocated Chunked	Algorithm 1	Algorithm 2
Qwen2.5 (7B)	Conversation	1.000	0.759	1.349	**0.787**	**0.878**
Qwen2.5 (7B)	Code	1.000	0.840	1.259	**0.525**	**0.582**
Llama 3.1 (8B)	Conversation	1.000	0.738	1.239	**0.758**	**0.839**
Llama 3.1 (8B)	Code	1.000	0.835	1.227	**0.510**	**0.565**

Algorithms 1 and 2, as well as the vLLM (Kwon et al., 2023) P/D baseline, assign one GPU to prefill and the second to decoding. Colocated runs a single vLLM instance using tensor parallelism across both GPUs, so both GPUs execute prefill and decoding. Colocated Chunked uses the same configuration with chunked prefill enabled.

Results. Both algorithms reduce TCT relative to P/D for each model and workload shown in Table 2. The reductions range from 21.3% to 49.0% for Algorithm 1 and from 12.2% to 43.5% for Algorithm 2. Both achieve larger reductions on the code workloads. Additional memory capacities, prefill and decode demands, and batch sizes are evaluated in Appendix H.

6 Related Work

LLM Inference Serving and KV Cache Management. Orca (Yu et al., 2022) introduces iteration-level scheduling, vLLM (Kwon et al., 2023) manages KV caches with PagedAttention, and Sarathi (Agrawal et al., 2023) uses chunked prefills. FastServe (Wu et al., 2023) uses preemptive scheduling, AlpaServe (Li et al., 2023) multiplexes models through model parallelism, and FlexGen (Sheng et al., 2023) offloads model state. DeepSpeed-Inference (Aminabadi et al., 2022) optimizes parallel execution, while DeepSpeed-FastGen (Holmes et al., 2024) uses Dynamic SplitFuse. SGLang (Zheng et al., 2024), Preble (Srivatsa et al., 2025), and CacheGen (Liu et al., 2024) address prefix reuse, distributed scheduling, and KV cache compression and streaming, respectively. These systems do not model joint admission under separate prefill and decode memory capacities.

Prefill-Decode Disaggregated Serving. Splitwise (Patel et al., 2024) separately provisions prefill and decoding, DistServe (Zhong et al., 2024) optimizes phase-specific parallelism, and TetriInfer (Hu et al., 2024b) combines disaggregation with prompt chunking. Mooncake (Qin et al., 2026) and MemServe (Hu et al., 2024a) provide disaggregated KV cache storage and reuse; SmartGen (Luo et al., 2026) studies selective cache transfer, and HexGen-2 (Jiang et al., 2025) targets heterogeneous resources. TaiChi (Wang et al., 2025a), Arrow (Wu et al., 2025), and P/D-Serve (Jin et al., 2024) adapt serving decisions to workloads and system conditions. These works do not provide approximation guarantees for joint admission scheduling.

Theoretical Scheduling Models. Jaillet et al. (2025) and Ao et al. (2025) study online scheduling under a unified memory constraint, while Wang et al. (2025b) consider heterogeneous prefill and decode lengths under a shared cache capacity. Feng et al. (2026b) derive competitive and constant-factor approximation guarantees for non-clairvoyant and clairvoyant scheduling, respectively, under a single KV cache capacity. Their subsequent work develops a general non-clairvoyant scheduling framework (Feng et al., 2026a). Chen et al. (2025) consider uncertain output lengths. Bari et al. (2025) study routing and throughput-oriented scheduling, while Dai et al. (2025) analyze the throughput optimality of work-conserving scheduling. These models do not capture retained prompt caches across separate prefill and decode memory pools.

7 Conclusion and Future Work

We formulated DMRS for disaggregated LLM inference, jointly scheduling prefill and decode admissions under separate memory capacities while accounting for prompt KV caches retained in prefill memory and growing decode KV caches. We proved strong NP-hardness, derived an LP lower bound, and developed two deterministic polynomial-time approximation algorithms. The first achieves an approximation ratio of $8+\frac{4}{1-\alpha}+\frac{8}{1-\beta}$ for $\alpha, \beta < 1$, while the second provides a uniform 36-approximation for every feasible instance, including instances with $\alpha = 1$ or $\beta = 1$.

The CPU experiments show that both algorithms perform well relative to exact ILP solutions and LP lower bounds. Experiments on two H100 GPUs show that both algorithms reduce total completion time (TCT) relative to vLLM P/D across all tested settings on code workloads.

Limitations. We assume that output token counts are known in advance, consistent with the theoretical settings studied by Wang et al. (2025b) and Jaillet et al. (2025). This assumption allows us to focus on the coupling between prefill and decode admissions under separate memory capacities. The theoretical model assumes instantaneous KV transfer, whereas the completion times measured in our GPU experiments include KV transfer overhead.

Future Work. We can extend DMRS in three directions. An online setting could address requests arriving over time and establish competitive guarantees. A non-clairvoyant setting could remove the assumption that output lengths are known in advance. A deadline-constrained setting could maximize the total weight of requests completed by their deadlines.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, LaTeX organization, literature search and reference formatting, and the review and presentation of the theoretical formulation, algorithms, mathematical claims, proofs, experimental methodology, and results. The authors independently reviewed and verified all AI-assisted content, including the cited sources, derivations, algorithms, proofs, experimental configurations, code, and reported results. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The model assumptions and algorithms are stated in Sections 2–4. Complete proofs of all theoretical claims, the notation table, and an exact time-indexed ILP formulation are provided in the appendix. The supplementary CPU and GPU artifacts provide the fixed configurations, trace processing procedures, implementation and validation code, frozen results, and scripts used to reproduce the reported experimental summaries.

REFERENCES

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient LLM inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. DeepSpeed-Inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15. IEEE, 2022.

Ruicheng Ao, Gan Luo, David Simchi-Levi, and Xinshang Wang. Optimizing LLM inference: Fluid-guided online scheduling with memory constraints. *arXiv preprint arXiv:2504.11320*, 2025.

Agrim Bari, Parikshit Hegde, and Gustavo de Veciana. Optimal scheduling algorithms for LLM inference: Theory and practice. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 9(3):1–43, 2025.

Zixi Chen, Yinyu Ye, and Zijie Zhou. Adaptively robust LLM inference optimization under prediction uncertainty. *arXiv preprint arXiv:2508.14544*, 2025.

JG Dai, Tianze Deng, Yueying Li, and Tianyi Peng. Throughput-optimal scheduling algorithms for LLM inference and AI agents. *arXiv preprint arXiv:2504.07347*, 2025.

Yiding Feng, Siyu Liu, Zonghan Yang, and Yuhao Zhang. General non-clairvoyant KV-cache scheduling via regime-aware routing. *arXiv preprint arXiv:2607.09248*, 2026a.

Yiding Feng, Zonghan Yang, and Yuhao Zhang. Competitive non-clairvoyant KV-cache scheduling for LLM inference. *arXiv preprint arXiv:2601.22996*, 2026b.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1993.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. DeepSpeed-FastGen: High-throughput text generation for LLMs via MII and DeepSpeed-Inference. *arXiv preprint arXiv:2401.08671*, 2024.

Ke Hong, Lufang Chen, Zhong Wang, Xiuhong Li, Qiuli Mao, Jianping Ma, Chao Xiong, Guanyu Wu, Buhe Han, Guohao Dai, et al. Semi-PD: Towards efficient LLM serving via phase-wise disaggregated computation and unified storage. *arXiv preprint arXiv:2504.19867*, 2025.

Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, et al. MemServe: Context caching for disaggregated LLM serving with elastic memory pool. *arXiv preprint arXiv:2406.17565*, 2024a.

Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, et al. Inference without interference: Disaggregate LLM inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024b.

Patrick Jaillet, Jiashuo Jiang, Konstantina Mellou, Marco Molinaro, Chara Podimata, and Zijie Zhou. Online scheduling for LLM inference with KV cache constraints. *arXiv preprint arXiv:2502.07115*, 2025.

Youhe Jiang, Ran Yan, and Binhang Yuan. HexGen-2: Disaggregated generative inference of LLMs in heterogeneous environment. In *International Conference on Learning Representations*, volume 2025, pp. 3985–4002, 2025.

Yibo Jin, Tao Wang, Huimin Lin, Mingyang Song, Peiyang Li, Yipeng Ma, Yicheng Shan, Zhengfan Yuan, Cailong Li, Yajing Sun, et al. P/D-Serve: Serving disaggregated large language model at scale. *arXiv preprint arXiv:2408.08147*, 2024.

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*, pp. 611–626, 2023.

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

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024.

Xuchuan Luo, Jiacheng Shen, Xin Wang, and Yangfan Zhou. SmartGen: Seamless disaggregated LLM inference with selective KV cache transfer. *arXiv preprint arXiv:2607.28150*, 2026.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A KVCache-centric disaggregated architecture for LLM serving. *ACM Transactions on Storage*, 22(4):1–38, 2026.

Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *International conference on learning representations*, volume 2025, pp. 37057–37082, 2025.

Chao Wang, Pengfei Zuo, Zhangyu Chen, Yunkai Liang, Zhou Yu, and Ming-Chang Yang. Prefill-decode aggregation or disaggregation? unifying both for goodput-optimized LLM serving. *arXiv preprint arXiv:2508.01989*, 2025a.

Meixuan Wang, Yinyu Ye, and Zijie Zhou. LLM serving optimization with variable prefill and decode lengths. *arXiv preprint arXiv:2508.06133*, 2025b.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.

Yu Wu, Tongxuan Liu, Yuting Zeng, Siyu Wu, Jun Xiong, Xianzhe Dong, Hailong Yang, Ke Zhang, and Jing Li. Arrow: Adaptive scheduling mechanisms for disaggregated LLM inference architecture. *arXiv preprint arXiv:2505.11916*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024.

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)*, pp. 521–538, 2022.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. SGLang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 193–210, 2024.

CONTENTS

1 Introduction **1**

2 System Model and Formulation **2**

2.1 System Model 2

2.2 DMRS Formulation 3

2.3 LP Relaxation 4

3 Approximation for Restricted Instances **5**

3.1 Algorithm Design 5

3.2 Approximation Guarantee 5

4 Approximation for General Instances **6**

4.1 Request Classification 6

4.2 Algorithm Design 6

4.3 Approximation Guarantee 7

5 Experiments **8**

5.1 CPU Evaluation 8

5.2 GPU Evaluation 8

6 Related Work **9**

7 Conclusion and Future Work **9**

References **11**

A Notation **15**

B Proof of Strong NP-Hardness **15**

C Proofs for the LP Relaxation **16**

D Detailed Algorithm Descriptions **17**

E Proofs for Algorithm 1 **18**

F Proofs for Algorithm 2 **20**

G Exact ILP Formulation **22**

H Additional Experimental Results **23**

A NOTATION

The notation used throughout the paper is summarized in Table 3.

Table 3: Summary of notation.

Symbol	Definition
$\mathcal{I} = \{1, \dots, n\}$	The set of LLM inference requests
m_i	The effective prefill processing time of request i
p_i	The number of input tokens (i.e., prefill memory demand) of request i
d_i	The number of output tokens (i.e., decode iterations) of request i
$h_i = p_i + d_i$	The peak decode memory demand of request i
$r_i(u) = p_i + u$	The decode memory demand after request i generates u output tokens
M_P, M_D	The prefill and decode memory capacities
S_i^P, S_i^D	The prefill and decode admission times of request i
C_i^P, C_i	The prefill and request completion times of request i
$\mathcal{U}(t)$	The set of requests not yet admitted to prefill during slot $[t, t+1)$
$\mathcal{E}(t)$	The set of requests executing prefill during slot $[t, t+1)$
$\mathcal{W}(t)$	The set of requests waiting for decode admission during slot $[t, t+1)$
$\mathcal{D}(t)$	The set of requests executing decoding during slot $[t, t+1)$
ℓ_i^P, ℓ_i^D	The mandatory prefill and decode memory-time loads of request i
α, β	The maximum normalized prefill and peak decode memory demands, respectively
$\bar{C}_i$	The relaxed completion-time variable of request i
$\mathrm{LP}^\star, \mathrm{OPT}$	The optimal objective values of $(\mathbf{LP})$ and $(\mathbf{O})$, respectively
$\mathcal{G}_q, \mathcal{G}_{\le k}$	The dyadic group and prefix in Algorithm 1
$\mathcal{L}_P, \mathcal{S}_P$	The prefill-large and prefill-small request sets
$\mathcal{L}_D, \mathcal{S}_D$	The decode-large and decode-small request sets
$\mathcal{C}^{LL}, \mathcal{C}^{SL}, \mathcal{C}^{LS}, \mathcal{C}^{SS}$	The four memory-demand classes
ρ, B_q	The geometric grouping ratio ρ and boundary $B_q = \rho^q$
$\mathcal{H}_q, \mathcal{H}_{\le k}$	The geometric group and prefix in Algorithm 2

B PROOF OF STRONG NP-HARDNESS

Proof. We reduce from the strongly NP-complete 3-PARTITION problem (Garey & Johnson, 1979). Given 3ν positive integers $a_1, \dots, a_{3\nu}$ satisfying $\sum_{i=1}^{3\nu} a_i = \nu B$ and $B/4 < a_i < B/2$ for every i, 3-PARTITION asks whether they can be partitioned into ν disjoint triples, each summing to B.

Construct a DMRS instance with $\mathcal{I} = \{1, \dots, 3\nu\}$. For each a_i, define a request i with $m_i = d_i = 1$ and $p_i = a_i$. Set $M_P = B$, $M_D = \sum_{i=1}^{3\nu}(a_i + 1) = \nu B + 3\nu$, and $K = 3\nu(\nu+3)/2$. Since $\sum_{i\in\mathcal{I}} h_i = M_D$, the decode memory constraint never prevents decode admission.

Suppose that the 3-PARTITION instance has a valid partition, with its triples indexed by $r = 1, \dots, \nu$. Admit the three requests in triple r to prefill at time $r-1$ and to decoding at time r. Their total prefill memory demand is B. At time r, their decode admission releases this prefill memory, allowing the next triple, if any, to be admitted to prefill at the same time. The requests in triple r complete at time $r+1$. Hence, the resulting schedule is feasible and has objective value $\sum_{i\in\mathcal{I}} C_i = 3\sum_{r=1}^{\nu}(r+1) = 3\nu(\nu+3)/2 = K$.

Conversely, suppose that the constructed DMRS instance has a feasible schedule satisfying $\sum_{i\in\mathcal{I}} C_i \le K$. Let $C_{(1)} \le \cdots \le C_{(3\nu)}$ denote the completion times in nondecreasing order. Since $p_i > B/4$, at most three requests can be admitted to prefill at any slot boundary. The first j completed requests must have been admitted to prefill by time $C_{(j)} - 2$, because both phases require one slot. Their prefill admission times therefore lie in $\{0, \dots, C_{(j)} - 2\}$, which contains $C_{(j)} - 1$ slot boundaries. Hence, $j \le 3(C_{(j)} - 1)$, so $C_{(j)} \ge 1 + \lceil j/3 \rceil$. Consequently,

$$\sum_{i\in\mathcal{I}} C_i = \sum_{j=1}^{3\nu} C_{(j)} \ge \sum_{j=1}^{3\nu} \left(1 + \left\lceil \frac{j}{3} \right\rceil\right) = K.$$

Thus, equality holds throughout, and in particular $C_{(3\nu)} = \nu + 1$. Since $m_i = d_i = 1$, every request satisfies $S_i^P \le C_i - 2 \le C_{(3\nu)} - 2 = \nu - 1$.

For $t = 0, \ldots, \nu - 1$, define $\mathcal{A}_t = \{i \in \mathcal{I} : S_i^P = t\}$. These sets partition $\mathcal{I}$, and prefill memory feasibility gives $\sum_{i \in \mathcal{A}_t} p_i \leq B$. Moreover,

$$\sum_{t=0}^{\nu-1} \sum_{i \in \mathcal{A}_t} p_i = \sum_{i \in \mathcal{I}} p_i = \nu B.$$

Therefore, $\sum_{i \in \mathcal{A}_t} p_i = B$ for every $t = 0, \ldots, \nu - 1$. Since $B/4 < p_i < B/2$, each $\mathcal{A}_t$ contains exactly three requests. The sets $\mathcal{A}_0, \ldots, \mathcal{A}_{\nu-1}$ therefore yield a valid 3-partition.

The reduction is computable in polynomial time, and its numerical parameters remain polynomially bounded under the strongly NP-complete restriction of 3-PARTITION. Therefore, DMRS is strongly NP-hard. □

C Proofs for the LP Relaxation

C.1 Proof of Lemma 1

Proof. Constraint (2b) follows from $C_i = S_i^D + d_i \geq S_i^P + m_i + d_i \geq m_i + d_i$, where the two inequalities follow from $S_i^D \geq S_i^P + m_i$ and $S_i^P \geq 0$, respectively.

Consider a nonempty subset $\mathcal{S} \subseteq \mathcal{I}$ and either memory pool. Let M denote the capacity of the selected pool, and let ℓ_i denote the corresponding mandatory memory-time load. Reindex the requests in $\mathcal{S}$ as $1, \ldots, s$, where $s = |\mathcal{S}|$, such that $C_1 \leq \cdots \leq C_s$. For each $j = 1, \ldots, s$, define $L_j = \sum_{r=1}^{j} \ell_r$.

For the prefill pool, the mandatory load of request r is completed by $C_r^P \leq C_r$. For the decode pool, its mandatory load is completed by C_r. Therefore, by time C_j, the selected pool must have accommodated the total load L_j of requests $1, \ldots, j$. Since the pool has capacity M in each slot, it can accommodate at most MC_j units of memory-time load by that time. Hence, $MC_j \geq L_j$.

Dividing this inequality by M, multiplying it by ℓ_j, and summing over $j = 1, \ldots, s$ gives

$$\sum_{j=1}^{s} \ell_j C_j \geq \frac{1}{M} \sum_{j=1}^{s} \ell_j L_j = \frac{1}{2M} \left[\left(\sum_{j=1}^{s} \ell_j \right)^2 + \sum_{j=1}^{s} \ell_j^2 \right].$$

The equality follows from $L_j = \sum_{r=1}^{j} \ell_r$.

Taking $(\ell_i, M) = (\ell_i^P, M_P)$ proves Constraint (2c), while taking $(\ell_i, M) = (\ell_i^D, M_D)$ proves Constraint (2d). Thus, every feasible schedule of $(\mathbf{O})$ induces a feasible solution to $(\mathbf{LP})$ with $\bar{C}_i = C_i$ for all $i \in \mathcal{I}$. Therefore, $\mathrm{LP}^\star \leq \mathrm{OPT}$. □

C.2 Proof of Lemma 2

Proof. Although $(\mathbf{LP})$ contains exponentially many subset constraints, they admit a polynomial-time separation oracle. Given a candidate vector $\bar{\mathbf{C}} = (\bar{C}_i)_{i \in \mathcal{I}}$, consider either memory pool. Let M denote its capacity and ℓ_i the corresponding mandatory memory-time load. For each $\mathcal{S} \subseteq \mathcal{I}$, define

$$F(\mathcal{S}) := 2M \sum_{i \in \mathcal{S}} \ell_i \bar{C}_i - \left(\sum_{i \in \mathcal{S}} \ell_i \right)^2 - \sum_{i \in \mathcal{S}} \ell_i^2.$$

For every nonempty $\mathcal{S}$, the corresponding subset constraint is violated if and only if $F(\mathcal{S}) < 0$.

An exact separation oracle follows by sorting. Set $a_i = M\bar{C}_i - \ell_i/2$ and $L(\mathcal{S}) = \sum_{i \in \mathcal{S}} \ell_i$. Then $F(\mathcal{S})/2 = \sum_{i \in \mathcal{S}} \ell_i a_i - L(\mathcal{S})^2/2$. Let $\mathcal{S}^\star$ minimize F over all subsets and write $L = L(\mathcal{S}^\star)$. Removing any $i \in \mathcal{S}^\star$ cannot improve the minimum, so $a_i \leq L - \ell_i/2$. Likewise, adding any $j \notin \mathcal{S}^\star$ cannot improve it, so $a_j \geq L + \ell_j/2$. Since all loads are positive, every member of $\mathcal{S}^\star$ precedes every nonmember when requests are sorted by a_i. Thus, some minimizing subset is a prefix of this order. Sorting and checking all prefixes, including the empty prefix with value zero,

takes $O(n \log n)$ time per pool. A negative minimum identifies a violated subset constraint; a zero minimum certifies all subset constraints for that pool. Checking both pools and Constraint (2b) gives a polynomial-time separation oracle.

It remains to bound the search region. Let $T_0 := \sum_{i \in \mathcal{I}} (m_i + d_i)$. Consider a schedule that processes the requests one at a time and completes both phases of each request before starting the next. By the feasibility conditions stated in Section 2.1, this schedule is feasible, and every request completes by time T_0. By Lemma 1, the corresponding completion-time vector is feasible for $(\mathbf{LP})$ and has objective value at most nT_0. Therefore, $\mathrm{LP}^\star \leq nT_0$. Since $\bar{C}_i \geq m_i + d_i > 0$, every optimal solution satisfies

$$0 \leq \bar{C}_i \leq nT_0, \qquad i \in \mathcal{I}.$$

These bounds have encoding length polynomial in the input size and may be added to $(\mathbf{LP})$ without changing its optimal value. They can also be separated directly. The equivalence of optimization and separation therefore implies that $(\mathbf{LP})$ can be solved in polynomial time using the ellipsoid method (Grötschel et al., 1993). □

LP Implementation. In the CPU experiments, we solve $(\mathbf{LP})$ using Gurobi and an iterative cutting plane procedure. We first solve the LP without the subset constraints. We then apply the exact prefix separator to both memory pools. At each iteration, we add up to 100 previously unseen prefix constraints with the largest violations for each pool. We solve the updated LP and repeat this process until no checked prefix constraint has a relative violation exceeding 10^{-8}.

For the GPU experiments, we use HiGHS to solve $(\mathbf{LP})$. The initial LP includes the subset constraints for each individual request and for the full request set in both memory pools. We then add violated prefix constraints iteratively.

D Detailed Algorithm Descriptions

D.1 Description of Algorithm 1

Line 1 solves the LP relaxation, obtains an optimal solution $\{\bar{C}_i\}_{i \in \mathcal{I}}$, and constructs the nonempty dyadic groups. Line 2 initializes the current time and the active request sets.

Lines 3–24 process the nonempty groups in increasing order of q. For each group $\mathcal{G}_q$, Line 4 initializes $\mathcal{U}_q = \mathcal{G}_q$. Lines 5–23 process the current group until $\mathcal{U}_q$, $\mathcal{E}$, $\mathcal{W}$, and $\mathcal{D}$ are all empty. At each event time t, Line 6 removes the requests that have completed decoding from $\mathcal{D}$, and Line 7 moves the requests that have completed prefill from $\mathcal{E}$ to $\mathcal{W}$.

Lines 8–13 implement the decode admission rule. Line 8 creates a copy $\widehat{\mathcal{W}}$ of the current waiting set. Lines 9–13 consider the requests in $\widehat{\mathcal{W}}$ in nondecreasing order of $m_i + d_i$, with ties broken by request index. Line 10 checks the peak reservation condition: request i is admitted only if its peak demand h_i, together with the peak demands h_j reserved by all active requests $j \in \mathcal{D}$, does not exceed M_D. Because each request remains in $\mathcal{D}$ until its decode completion event at Line 6, its peak demand remains reserved throughout decoding. Upon admission, Line 11 sets $S_i^D = t$, removes i from $\mathcal{W}$, and adds it to $\mathcal{D}$.

Lines 14–19 implement the prefill admission rule. Line 14 creates a copy $\widehat{\mathcal{U}}_q$ of the current unstarted set. Lines 15–19 consider the requests in $\widehat{\mathcal{U}}_q$ in nondecreasing order of $m_i + d_i$, with ties broken by request index. A request i is admitted to prefill if its demand p_i, together with the prefill memory occupied by the requests in $\mathcal{E}$ and $\mathcal{W}$, does not exceed M_P. Upon admission, Line 17 sets $S_i^P = t$, removes i from $\mathcal{U}_q$, and adds it to $\mathcal{E}$.

If the current group remains unfinished after both admission rules, Lines 20–22 advance t to the next prefill or decode completion time.

D.2 Description of Algorithm 2

Line 1 solves $(\mathbf{LP})$, obtains an optimal solution, and constructs the four request classes. Line 2 forms the nonempty geometric groups, and Line 3 sets $t = 0$ and initializes $\mathcal{E}$, $\mathcal{W}$, and $\mathcal{D}$ as empty sets.

Line 4 processes the nonempty groups in increasing order of q. Lines 5–7 process the requests in $\mathcal{H}_q \cap \mathcal{C}^{LL}$ one at a time, in nondecreasing order of $m_i + d_i$. For each request, Line 6 sets its admission times and advances t to its completion time.

Lines 8–29 process the remaining classes in the order $\mathcal{C}^{SL}$, $\mathcal{C}^{LS}$, and $\mathcal{C}^{SS}$. Line 9 collects the unstarted requests in the current group and class, and the loop in Lines 10–28 continues until all of them have completed. At each event time, Lines 11–12 remove completed decode requests and move completed prefill requests to $\mathcal{W}$.

Lines 13–18 implement the decode admission rule. Line 13 creates a copy $\widehat{\mathcal{W}}$ of the current waiting set. Lines 14–18 consider the requests in $\widehat{\mathcal{W}}$ in nondecreasing order of $m_i + d_i$, with ties broken by request index. Line 15 checks the peak reservation condition: request i is admitted only if $\sum_{j\in\mathcal{D}} h_j + h_i \le M_D$. Because each request remains in $\mathcal{D}$ until its decode completion event at Line 11, its peak demand remains reserved throughout decoding. Upon admission, Line 16 sets its decode admission time and moves it from $\mathcal{W}$ to $\mathcal{D}$.

Lines 19–24 implement the prefill admission rule. Line 19 creates a copy $\widehat{\mathcal{U}}_{q,\mathcal{C}}$ of the unstarted requests in the current group and class. Lines 20–24 consider these requests in nondecreasing order of $m_i + d_i$, with ties broken by request index. A request is admitted when its prefill memory demand can be accommodated together with the memory occupied by the requests in $\mathcal{E}$ and $\mathcal{W}$. Upon admission, Line 22 sets its prefill admission time and moves it from $\mathcal{U}_{q,\mathcal{C}}$ to $\mathcal{E}$.

If the current class remains unfinished, Lines 25–27 advance t to the next prefill or decode completion time.

E Proofs for Algorithm 1

We first bound the cumulative memory-time loads of the dyadic prefixes defined in the proof of Theorem 2.

Lemma 3. *For each $k \in \mathbb{Z}$, the total prefill and decode memory-time loads of the prefix $\mathcal{G}_{\le k}$ satisfy $\sum_{i\in\mathcal{G}_{\le k}} \ell_i^P \le 2M_P\, 2^k$ and $\sum_{i\in\mathcal{G}_{\le k}} \ell_i^D \le 2M_D\, 2^k$, respectively.*

Proof. If $\mathcal{G}_{\le k} = \emptyset$, the claim is immediate. Otherwise, each $i \in \mathcal{G}_{\le k}$ belongs to some $\mathcal{G}_q$ with $q \le k$, and hence $\bar{C}_i \le 2^q \le 2^k$. Therefore,

$$\sum_{i\in\mathcal{G}_{\le k}} \ell_i^P \bar{C}_i \le 2^k \sum_{i\in\mathcal{G}_{\le k}} \ell_i^P.$$

Applying Constraint (2c) to $\mathcal{G}_{\le k}$ and dropping the nonnegative term $\sum_{i\in\mathcal{G}_{\le k}} (\ell_i^P)^2$ gives

$$\sum_{i\in\mathcal{G}_{\le k}} \ell_i^P \bar{C}_i \ge \frac{1}{2M_P} \left(\sum_{i\in\mathcal{G}_{\le k}} \ell_i^P \right)^2.$$

Combining the two inequalities yields

$$\sum_{i\in\mathcal{G}_{\le k}} \ell_i^P \le 2M_P\, 2^k.$$

Applying the same argument to Constraint (2d) gives

$$\sum_{i\in\mathcal{G}_{\le k}} \ell_i^D \le 2M_D\, 2^k,$$

which completes the proof. □

Algorithm 1 reserves the peak decode memory demand h_i throughout the d_i decode slots of request i. Lemma 4 shows that this reserved memory-time load is less than twice the mandatory decode memory-time load captured by the LP.

Lemma 4. *For each request $i \in \mathcal{I}$, its reserved decode memory-time $h_i d_i$ satisfies $\ell_i^D \le h_i d_i < 2\ell_i^D$. Consequently, for every $k \in \mathbb{Z}$, $\sum_{i \in \mathcal{G}_{\le k}} h_i d_i < 4M_D\, 2^k$.*

Proof. Using $h_i = p_i + d_i$ and $\ell_i^D = p_i d_i + d_i(d_i+1)/2$, we have

$$h_i d_i - \ell_i^D = \frac{d_i(d_i-1)}{2} \ge 0, \qquad 2\ell_i^D - h_i d_i = d_i(p_i+1) > 0.$$

Hence, $\ell_i^D \le h_i d_i < 2\ell_i^D$. Summing over $\mathcal{G}_{\le k}$ and applying Lemma 3 gives

$$\sum_{i \in \mathcal{G}_{\le k}} h_i d_i < 2 \sum_{i \in \mathcal{G}_{\le k}} \ell_i^D \le 4M_D\, 2^k,$$

which completes the proof. □

Lemma 5. *For each $k \in \mathbb{Z}$, Algorithm 1 completes all requests in $\mathcal{G}_{\le k}$ by time*

$$\left(4 + \frac{2}{1-\alpha} + \frac{4}{1-\beta}\right) 2^k.$$

Proof. If $\mathcal{G}_{\le k} = \emptyset$, the claim is immediate. Otherwise, let T_k denote the time at which all requests in $\mathcal{G}_{\le k}$ have completed. By the strict group barrier, only requests in $\mathcal{G}_{\le k}$ are processed during $[0, T_k)$.

For each slot $t < T_k$, let $\mathcal{U}_q(t)$, $\mathcal{E}(t)$, $\mathcal{W}(t)$, and $\mathcal{D}(t)$ denote the request sets during $[t, t+1)$, where $\mathcal{G}_q$ is the group currently being processed. Define

$$\begin{aligned}
\mathcal{T}_1 &:= \{t < T_k : \mathcal{W}(t) \neq \emptyset\},\\
\mathcal{T}_2 &:= \{t < T_k : \mathcal{W}(t) = \emptyset,\, \mathcal{U}_q(t) \neq \emptyset\},\\
\mathcal{T}_3 &:= \{t < T_k : \mathcal{W}(t) = \mathcal{U}_q(t) = \emptyset,\, \mathcal{E}(t) \neq \emptyset\},\\
\mathcal{T}_4 &:= \{t < T_k : \mathcal{W}(t) = \mathcal{U}_q(t) = \mathcal{E}(t) = \emptyset,\, \mathcal{D}(t) \neq \emptyset\}.
\end{aligned}$$

For every $t \in \{0, \dots, T_k - 1\}$, at least one of $\mathcal{U}_q(t)$, $\mathcal{E}(t)$, $\mathcal{W}(t)$, and $\mathcal{D}(t)$ is nonempty. Therefore, $\mathcal{T}_1, \dots, \mathcal{T}_4$ are pairwise disjoint and partition the slots in $[0, T_k)$. Here, $|\mathcal{T}_r|$ denotes the number of slots in $\mathcal{T}_r$, for $r = 1, \dots, 4$.

Case 1: $t \in \mathcal{T}_1$. For any $t \in \mathcal{T}_1$, choose a request $i \in \mathcal{W}(t)$ that remains waiting for decode admission. By the decode admission rule in Algorithm 1, $\sum_{j \in \mathcal{D}(t)} h_j + h_i > M_D$. Since $h_i \le \beta M_D$, it follows that

$$\sum_{j \in \mathcal{D}(t)} h_j > (1-\beta) M_D.$$

Summing this inequality over $t \in \mathcal{T}_1$ gives

$$|\mathcal{T}_1|(1-\beta)M_D < \sum_{t \in \mathcal{T}_1} \sum_{j \in \mathcal{D}(t)} h_j \le \sum_{i \in \mathcal{G}_{\le k}} h_i d_i < 4M_D\, 2^k,$$

where the final inequality follows from Lemma 4. Therefore,

$$|\mathcal{T}_1| < \frac{4}{1-\beta} 2^k.$$

Case 2: $t \in \mathcal{T}_2$. For any $t \in \mathcal{T}_2$, choose a request $i \in \mathcal{U}_q(t)$ that remains unadmitted to prefill. Because $\mathcal{W}(t) = \emptyset$, the prefill admission rule in Algorithm 1 implies $\sum_{j \in \mathcal{E}(t)} p_j + p_i > M_P$. Since $p_i \le \alpha M_P$, it follows that

$$\sum_{j \in \mathcal{E}(t)} p_j > (1-\alpha) M_P.$$

Summing this inequality over $t \in \mathcal{T}_2$ gives

$$|\mathcal{T}_2|(1-\alpha)M_P < \sum_{t \in \mathcal{T}_2} \sum_{j \in \mathcal{E}(t)} p_j \le \sum_{i \in \mathcal{G}_{\le k}} \ell_i^P \le 2M_P\, 2^k,$$

where the final inequality follows from Lemma 3. Therefore,

$$|\mathcal{T}_2| < \frac{2}{1-\alpha} 2^k.$$

Case 3: $t \in \mathcal{T}_3$. Fix a nonempty group $\mathcal{G}_q$. Suppose that $\mathcal{T}_3$ contains a slot during which $\mathcal{G}_q$ is being processed, and consider the first such slot. At this time, $\mathcal{U}_q(t) = \emptyset$, so no additional request from $\mathcal{G}_q$ can enter prefill. The group barrier also prevents requests from later groups from entering prefill. For every $i \in \mathcal{G}_q$, Constraint (2b) and the definition of $\mathcal{G}_q$ imply

$$m_i \le m_i + d_i \le \bar{C}_i \le 2^q.$$

Thus, every request executing prefill in this slot completes prefill within at most 2^q additional slots. Since no new request can enter prefill, the number of slots in $\mathcal{T}_3$ during which $\mathcal{G}_q$ is processed is at most 2^q.

Case 4: $t \in \mathcal{T}_4$. Fix a nonempty group $\mathcal{G}_q$. Suppose that $\mathcal{T}_4$ contains a slot during which $\mathcal{G}_q$ is being processed, and consider the first such slot. At this time, $\mathcal{U}_q(t) = \mathcal{W}(t) = \mathcal{E}(t) = \emptyset$, so only requests executing decoding remain. No additional request can enter decoding thereafter. For every $i \in \mathcal{G}_q$, Constraint (2b) and the definition of $\mathcal{G}_q$ imply

$$d_i \le m_i + d_i \le \bar{C}_i \le 2^q.$$

Thus, every remaining request completes decoding within at most 2^q additional slots. Therefore, the number of slots in $\mathcal{T}_4$ during which $\mathcal{G}_q$ is processed is at most 2^q.

Since

$$\sum_{\substack{q \le k \\ \mathcal{G}_q \neq \emptyset}} 2^q \le \sum_{q=-\infty}^{k} 2^q = 2^{k+1},$$

combining the four cases gives

$$\begin{aligned} T_k = |\mathcal{T}_1| + |\mathcal{T}_2| + |\mathcal{T}_3| + |\mathcal{T}_4| &< \frac{4}{1-\beta} 2^k + \frac{2}{1-\alpha} 2^k + 2 \sum_{\substack{q \le k \\ \mathcal{G}_q \neq \emptyset}} 2^q \\ &\le \frac{4}{1-\beta} 2^k + \frac{2}{1-\alpha} 2^k + 4 \cdot 2^k = \left(4 + \frac{2}{1-\alpha} + \frac{4}{1-\beta}\right) 2^k. \end{aligned}$$

This completes the proof. □

F Proofs for Algorithm 2

Slot Categories. To bound the completion time of a geometric prefix, we assign every execution slot, after the admission decisions at its starting time, to one charging category. During the $\mathcal{C}^{LL}$ phase of group $\mathcal{H}_q$, each prefill slot is assigned to the large prefill category and each decode slot to the large decode category. For each remaining class $\mathcal{C} \in \{\mathcal{C}^{SL}, \mathcal{C}^{LS}, \mathcal{C}^{SS}\}$, consider the phase that processes $\mathcal{H}_q \cap \mathcal{C}$. Let $\mathcal{U}_{q,\mathcal{C}}(t)$, $\mathcal{E}(t)$, $\mathcal{W}(t)$, and $\mathcal{D}(t)$ denote the request sets after admission at time t. The slots in these three class phases are assigned by the following priority rules.

- For $\mathcal{C}^{SL}$, assign the slot to the large decode category if $\mathcal{D}(t) \neq \emptyset$. Otherwise, assign it to the blocked small prefill category if $\mathcal{U}_{q,\mathcal{C}}(t) \neq \emptyset$, and to the prefill tail category in the remaining case.
- For $\mathcal{C}^{LS}$, assign the slot to the blocked small decode category if $\mathcal{W}(t) \neq \emptyset$. Otherwise, assign it to the large prefill category if $\mathcal{E}(t) \neq \emptyset$, and to the decode tail category in the remaining case.
- For $\mathcal{C}^{SS}$, assign the slot to the blocked small decode category if $\mathcal{W}(t) \neq \emptyset$. Otherwise, assign it to the blocked small prefill category if $\mathcal{U}_{q,\mathcal{C}}(t) \neq \emptyset$, to the prefill tail category if $\mathcal{E}(t) \neq \emptyset$, and to the decode tail category in the remaining case.

Under this assignment, a prefill tail slot satisfies $\mathcal{U}_{q,\mathcal{C}}(t) = \mathcal{W}(t) = \emptyset$ and $\mathcal{E}(t) \neq \emptyset$; requests in $\mathcal{D}(t)$ may execute concurrently. A decode tail slot satisfies $\mathcal{U}_{q,\mathcal{C}}(t) = \mathcal{E}(t) = \mathcal{W}(t) = \emptyset$ and $\mathcal{D}(t) \neq \emptyset$. These rules are exhaustive. In a $\mathcal{C}^{SL}$ phase, if $\mathcal{D}(t) = \emptyset$, then $\mathcal{W}(t) = \emptyset$, since any waiting request fits in an empty decode pool. If $\mathcal{U}_{q,\mathcal{C}}(t) = \emptyset$ as well, every unfinished request must be in $\mathcal{E}(t)$, which gives a prefill tail. In a $\mathcal{C}^{LS}$ phase, if $\mathcal{W}(t) = \mathcal{E}(t) = \emptyset$, then $\mathcal{U}_{q,\mathcal{C}}(t) = \emptyset$, since any unstarted request fits in an empty prefill pool. Every unfinished request must therefore be in $\mathcal{D}(t)$, which gives a decode tail. The four priority cases for $\mathcal{C}^{SS}$ directly cover every possible unfinished state. The strict group and class barriers ensure that no request outside the current phase affects this classification.

Lemma 6. *For every $k \in \mathbb{Z}$, the prefix $\mathcal{H}_{\leq k}$ satisfies $\sum_{i\in\mathcal{H}_{\leq k}} \ell_i^P \leq 2M_P B_k$ and $\sum_{i\in\mathcal{H}_{\leq k}} \ell_i^D \leq 2M_D B_k$.*

Proof. The claim is immediate when $\mathcal{H}_{\leq k} = \emptyset$. Otherwise, every $i \in \mathcal{H}_{\leq k}$ satisfies $\bar{C}_i \leq B_k$. Applying Constraint (2c) to $\mathcal{H}_{\leq k}$ gives

$$\frac{1}{2M_P}\left(\sum_{i\in\mathcal{H}_{\leq k}} \ell_i^P\right)^2 \leq \sum_{i\in\mathcal{H}_{\leq k}} \ell_i^P \bar{C}_i \leq B_k \sum_{i\in\mathcal{H}_{\leq k}} \ell_i^P.$$

Since the total prefill load is positive, division gives

$$\sum_{i\in\mathcal{H}_{\leq k}} \ell_i^P \leq 2M_P B_k.$$

Replacing (ℓ_i^P, M_P) with (ℓ_i^D, M_D) and applying Constraint (2d) proves the decode bound. □

Lemma 7. *For every $k \in \mathbb{Z}$, Algorithm 2 completes all requests in $\mathcal{H}_{\leq k}$ by time $\left(12 + \frac{4\rho}{\rho-1}\right) B_k$.*

Proof. The claim is immediate when $\mathcal{H}_{\leq k} = \emptyset$. Otherwise, define $T_k := \max_{i\in\mathcal{H}_{\leq k}} C_i$, the time at which the algorithm completes the prefix $\mathcal{H}_{\leq k}$. The strict group barrier implies that only requests in this prefix are processed during $[0, T_k)$.

Let $\mathcal{T}_{PL}$, $\mathcal{T}_{PS}$, $\mathcal{T}_{DL}$, and $\mathcal{T}_{DS}$ denote the sets of slots assigned to the large prefill, blocked small prefill, large decode, and blocked small decode categories, respectively. Let $\mathcal{T}_{\text{tail}}$ contain all prefill and decode tail slots.

1. Prefill Categories. For every $i \in \mathcal{L}_P$, $p_i > M_P/2$, and hence

$$m_i = \frac{\ell_i^P}{p_i} < \frac{2\ell_i^P}{M_P}.$$

Therefore,

$$|\mathcal{T}_{PL}| \leq \sum_{i\in\mathcal{H}_{\leq k}\cap\mathcal{L}_P} m_i \leq \frac{2}{M_P} \sum_{i\in\mathcal{H}_{\leq k}\cap\mathcal{L}_P} \ell_i^P.$$

For every $t \in \mathcal{T}_{PS}$, a prefill-small request remains unadmitted while $\mathcal{W}(t) = \emptyset$. If i is such a request, then $p_i \leq M_P/2$, and the failed prefill admission implies $\sum_{j\in\mathcal{E}(t)} p_j + p_i > M_P$. Consequently, $\sum_{j\in\mathcal{E}(t)} p_j > \frac{M_P}{2}$. All requests in $\mathcal{E}(t)$ belong to $\mathcal{S}_P$. Summing over $\mathcal{T}_{PS}$ gives

$$|\mathcal{T}_{PS}|\frac{M_P}{2} < \sum_{t\in\mathcal{T}_{PS}} \sum_{j\in\mathcal{E}(t)} p_j \leq \sum_{i\in\mathcal{H}_{\leq k}\cap\mathcal{S}_P} \ell_i^P.$$

Combining the two prefill bounds and applying Lemma 6 yields

$$|\mathcal{T}_{PL}| + |\mathcal{T}_{PS}| \leq \frac{2}{M_P} \sum_{i\in\mathcal{H}_{\leq k}} \ell_i^P \leq 4B_k.$$

2. Decode Categories. Under peak decode reservation, each admitted request i reserves h_i until its completion. Hence, no two decode-large requests can overlap in the decode pool. For every $i \in \mathcal{L}_D$, $h_i > M_D/2$. By Lemma 4, $h_i d_i < 2\ell_i^D$, and hence $d_i < \frac{2h_i d_i}{M_D} < \frac{4\ell_i^D}{M_D}$. Therefore,

$$|\mathcal{T}_{DL}| \le \sum_{i \in \mathcal{H}_{\le k} \cap \mathcal{L}_D} d_i \le \frac{4}{M_D} \sum_{i \in \mathcal{H}_{\le k} \cap \mathcal{L}_D} \ell_i^D.$$

For every $t \in \mathcal{T}_{DS}$, a decode-small request remains waiting after decode admission. If i is such a request, then $h_i \le M_D/2$, and the failed decode admission implies

$$\sum_{j \in \mathcal{D}(t)} h_j + h_i > M_D.$$

Consequently, $\sum_{j \in \mathcal{D}(t)} h_j > \frac{M_D}{2}$. All requests in $\mathcal{D}(t)$ belong to $\mathcal{S}_D$. Summing over $\mathcal{T}_{DS}$ and using $h_i d_i < 2\ell_i^D$ gives

$$|\mathcal{T}_{DS}| \frac{M_D}{2} < \sum_{t \in \mathcal{T}_{DS}} \sum_{j \in \mathcal{D}(t)} h_j \le \sum_{i \in \mathcal{H}_{\le k} \cap \mathcal{S}_D} h_i d_i < 2 \sum_{i \in \mathcal{H}_{\le k} \cap \mathcal{S}_D} \ell_i^D.$$

Combining the two decode bounds and applying Lemma 6 yields

$$|\mathcal{T}_{DL}| + |\mathcal{T}_{DS}| \le \frac{4}{M_D} \sum_{i \in \mathcal{H}_{\le k}} \ell_i^D \le 8B_k.$$

3. Tail Categories. For each nonempty group $\mathcal{H}_q$, tail slots can occur in four class phases: the prefill tail of $\mathcal{C}^{SL}$, the decode tail of $\mathcal{C}^{LS}$, and the prefill and decode tails of $\mathcal{C}^{SS}$. Every $i \in \mathcal{H}_q$ satisfies $m_i + d_i \le \bar{C}_i \le B_q$, and hence $m_i, d_i \le B_q$.

When a prefill tail first occurs in a class phase, no unstarted request remains, so no additional request can enter prefill during that phase. All requests currently executing prefill therefore complete prefill within at most B_q slots. When a decode tail first occurs, no request remains in $\mathcal{U}_{q,\mathcal{C}}(t)$, $\mathcal{E}(t)$, or $\mathcal{W}(t)$. Hence, no additional request can enter decoding, and all requests currently executing decoding complete within at most B_q slots. Therefore,

$$|\mathcal{T}_{\text{tail}}| \le 4 \sum_{\substack{q \le k \\ \mathcal{H}_q \ne \emptyset}} B_q \le 4 \sum_{q \le k} \rho^q = \frac{4\rho}{\rho - 1} B_k.$$

Because the slot categories partition $[0, T_k)$,

$$\begin{aligned} T_k &= |\mathcal{T}_{PL}| + |\mathcal{T}_{PS}| + |\mathcal{T}_{DL}| + |\mathcal{T}_{DS}| + |\mathcal{T}_{\text{tail}}| \\ &\le 4B_k + 8B_k + \frac{4\rho}{\rho - 1} B_k = \left(12 + \frac{4\rho}{\rho - 1}\right) B_k. \end{aligned}$$

This completes the proof. □

G Exact ILP Formulation

The following time-indexed ILP computes exact optimal solutions for every instance satisfying $p_i \le M_P$ and $h_i \le M_D$ for all $i \in \mathcal{I}$. Under these conditions, processing the requests sequentially, with each request entering decoding immediately after completing prefill, gives a feasible schedule with makespan

$$H := \sum_{i \in \mathcal{I}} (m_i + d_i).$$

An optimal schedule can be chosen with makespan at most H. Indeed, for any feasible schedule with makespan $C_{\max} := \max_{i \in \mathcal{I}} C_i$,

$$\sum_{t=0}^{C_{\max}-1} \big(|\mathcal{E}(t)| + |\mathcal{D}(t)|\big) = \sum_{i \in \mathcal{I}} (m_i + d_i) = H.$$

If $C_{\max} > H$, some slot contains no request executing prefill or decoding. Deleting this slot and shifting all subsequent events one slot earlier preserves precedence, does not increase memory usage in either pool, and strictly decreases at least one completion time, contradicting optimality.

Define the slot set $\mathbb{T} := \{0, \dots, H-1\}$ and the feasible prefill and decode admission-time sets $\mathbb{T}_i^P := \{0, \dots, H - m_i - d_i\}$ and $\mathbb{T}_i^D := \{0, \dots, H - d_i\}$, respectively. For each $i \in \mathcal{I}$, let $x_{i,s}^P$ and $x_{i,s}^D$ indicate whether request i is admitted to prefill and decoding at time s, respectively. The ILP is

$$
\begin{aligned}
(\mathbf{ILP}) \qquad \min \quad & \sum_{i \in \mathcal{I}} \sum_{s \in \mathbb{T}_i^D} (s + d_i) x_{i,s}^D && && \text{(3a)} \\
\text{s.t.} \quad & \sum_{s \in \mathbb{T}_i^P} x_{i,s}^P = 1, && \forall i \in \mathcal{I}, && \text{(3b)} \\
& \sum_{s \in \mathbb{T}_i^D} x_{i,s}^D = 1, && \forall i \in \mathcal{I}, && \text{(3c)} \\
& \sum_{s \in \mathbb{T}_i^D} s\, x_{i,s}^D \geq \sum_{s \in \mathbb{T}_i^P} (s + m_i) x_{i,s}^P, && \forall i \in \mathcal{I}, && \text{(3d)} \\
& \sum_{i \in \mathcal{I}} p_i \left(\sum_{\substack{s \in \mathbb{T}_i^P \\ s \leq t}} x_{i,s}^P - \sum_{\substack{s \in \mathbb{T}_i^D \\ s \leq t}} x_{i,s}^D \right) \leq M_P, && \forall t \in \mathbb{T}, && \text{(3e)} \\
& \sum_{i \in \mathcal{I}} \sum_{\substack{s \in \mathbb{T}_i^D \\ s \leq t < s + d_i}} (p_i + t - s + 1) x_{i,s}^D \leq M_D, && \forall t \in \mathbb{T}, && \text{(3f)} \\
& x_{i,s}^P \in \{0,1\}, && \forall i \in \mathcal{I},\ \forall s \in \mathbb{T}_i^P, && \text{(3g)} \\
& x_{i,s}^D \in \{0,1\}, && \forall i \in \mathcal{I},\ \forall s \in \mathbb{T}_i^D. && \text{(3h)}
\end{aligned}
$$

The objective minimizes the sum of request completion times. Constraints (3b) and (3c) assign one admission time to each phase, while Constraint (3d) enforces prefill-to-decode precedence. In Constraint (3e), the difference between the two cumulative sums equals one exactly when $S_i^P \leq t < S_i^D$, thereby enforcing the prefill memory capacity. Constraint (3f) accounts for the $p_i + t - s + 1$ units of decode memory used during slot $[t, t+1)$ by a request admitted to decoding at time s.

Theorem 4. *Problems* $(\mathbf{ILP})$ *and* $(\mathbf{O})$ *have the same optimal objective value.*

Proof. Consider an optimal schedule for $(\mathbf{O})$ with makespan at most H. The horizon bound and precedence relation imply $S_i^P \in \mathbb{T}_i^P$ and $S_i^D \in \mathbb{T}_i^D$. Setting $x_{i,S_i^P}^P = x_{i,S_i^D}^D = 1$ and all other variables to zero yields a feasible solution to $(\mathbf{ILP})$ with the same objective value.

Conversely, Constraints (3b) and (3c) define admission times S_i^P and S_i^D for every request. The remaining constraints enforce precedence and both memory capacities in every slot, while the objective equals $\sum_i (S_i^D + d_i) = \sum_i C_i$. Thus, every feasible ILP solution defines a feasible schedule for $(\mathbf{O})$ with the same objective value. □

H Additional Experimental Results

This appendix provides the complete CPU and GPU results supporting Section 5. In the CPU experiments, ALG/OPT denotes the algorithmic objective divided by the exact optimum obtained from the time-indexed ILP, whereas ALG/LP uses the LP lower bound $\mathrm{LP}^\star$. In the GPU experiments, total completion time (TCT) is normalized by the corresponding P/D baseline under the same model, workload, and operating condition.

H.1 CPU Results

We evaluate both algorithms on synthetic and trace-derived instances. Each synthetic setting uses 30 random seeds. For each request, we randomly select one of four sampling schemes with equal prob-

ability: uniform, bimodal, heavy-tailed, or correlated sampling. The correlated scheme generates m_i, p_i, d_i from a shared random variable. All three parameters lie in $\{1, \ldots, 20\}$. The supplementary code specifies the sampling rules.

For each trace workload, we use 30 windows of 250 requests and take the first n requests from each window. We set p_i and d_i to the recorded input and output token counts, respectively, and use $m_i = \lceil p_i/512 \rceil$. All CPU instances assume that requests are released at time zero.

For the capacity experiments, we set $M_P = \lceil \kappa \max_i p_i \rceil$ and $M_D = \lceil \kappa \max_i h_i \rceil$, where κ is the capacity multiplier.

Synthetic Workloads. Figure 2 compares both algorithms with the exact ILP optimum on small instances. ALG/OPT generally increases with n, and Algorithm 1 has the lower ratio across the tested capacities. For larger instances, Figure 3 shows that ALG/LP decreases as n grows, while LP solution time increases.

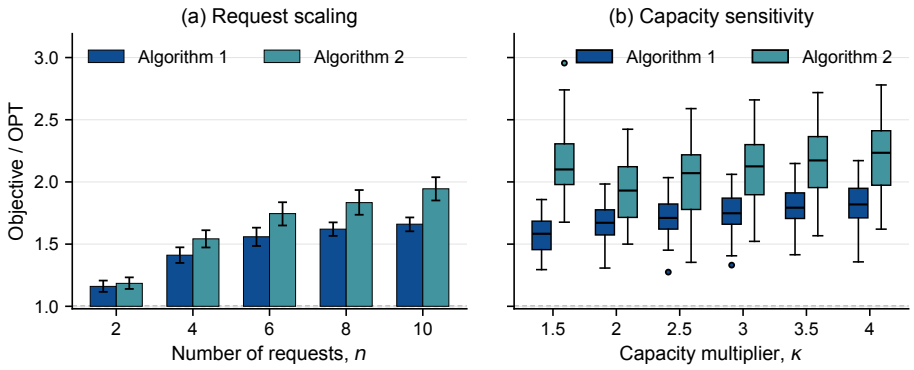


Figure 2: (a) ALG/OPT versus n at $\kappa = 2$. (b) ALG/OPT versus κ at $n = 10$.

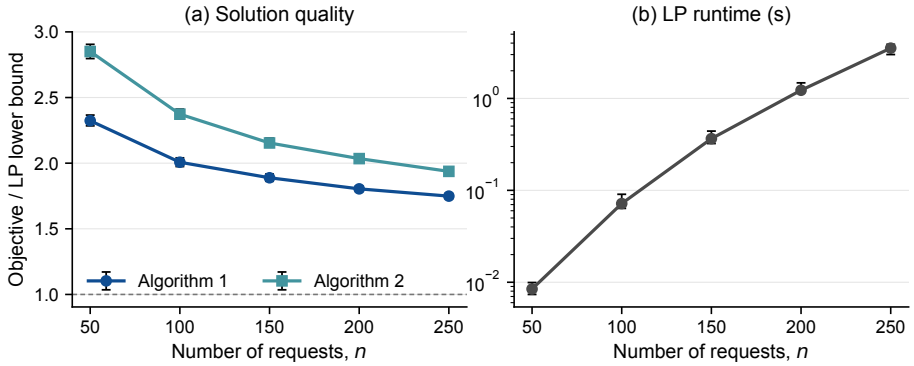


Figure 3: (a) ALG/LP versus n at $\kappa = 2$. (b) LP solution time versus n at $\kappa = 2$.

Trace-Derived Workloads. We use conversation and code workloads derived from the Azure LLM Inference Trace 2023 (Patel et al., 2024). Figure 4 shows that ALG/LP decreases with n for both workloads, with Algorithm 1 yielding lower ratios throughout.

Figures 5 and 6 show that ALG/LP generally increases with κ, while algorithm runtime increases with n. Runtime includes both LP solution and schedule construction. The two runtime curves nearly overlap because both algorithms use the same LP solution, which accounts for most of their runtime, especially for larger n.

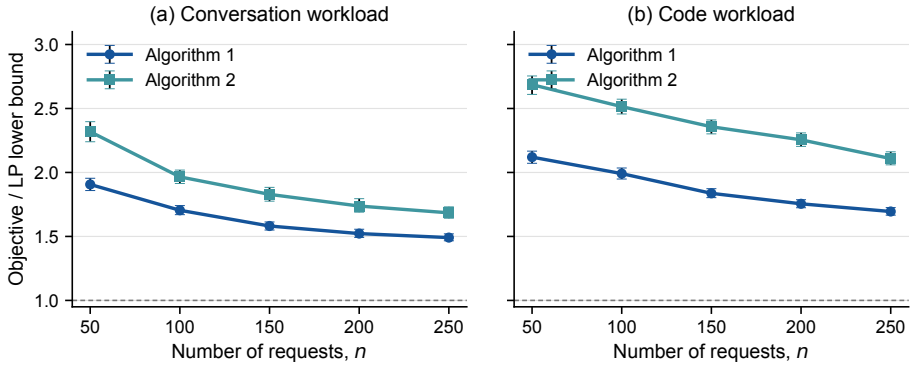


Figure 4: ALG/LP versus n at $\kappa = 2$ for (a) conversation and (b) code workloads.

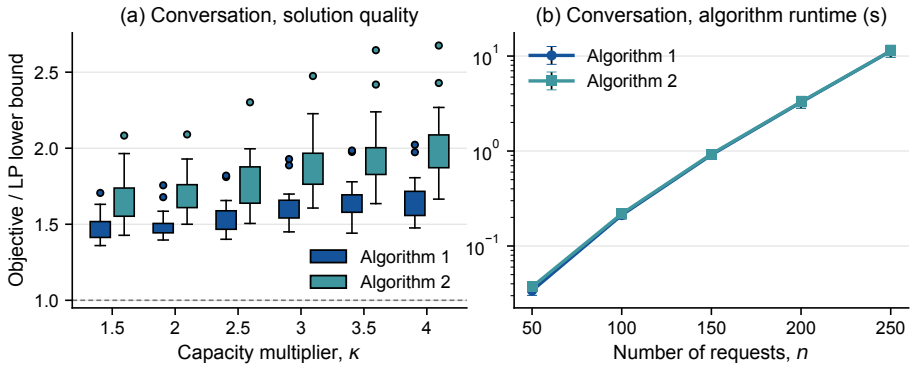


Figure 5: Conversation workload. (a) ALG/LP versus κ at $n = 250$. (b) Runtime versus n at $\kappa = 2$.

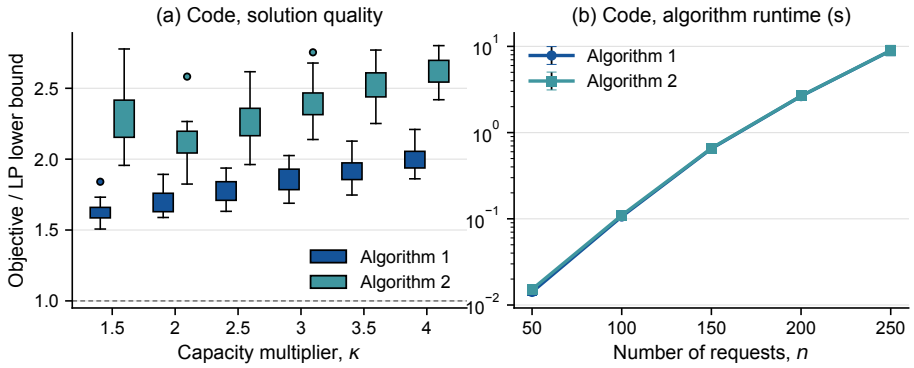


Figure 6: Code workload. (a) ALG/LP versus κ at $n = 250$. (b) Runtime versus n at $\kappa = 2$.

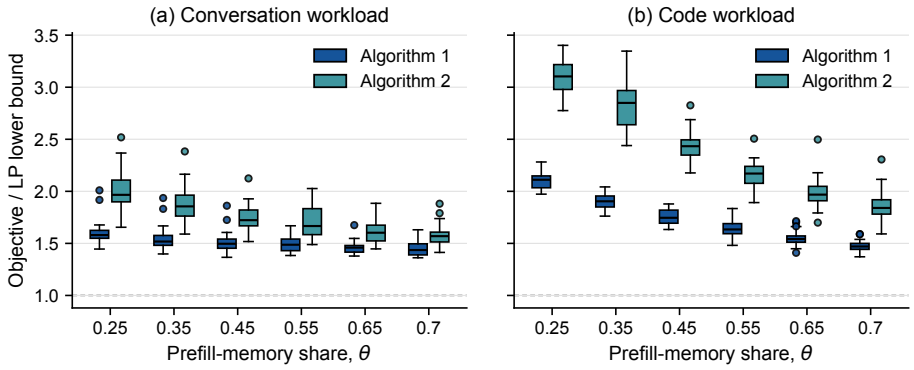


Figure 7: ALG/LP versus θ for (a) conversation and (b) code workloads.

We further examine how a fixed total memory budget is divided between the two pools. For each instance, let $B = \lceil 2 \max_i p_i \rceil + \lceil 2 \max_i h_i \rceil$ denote the total budget at $\kappa = 2$. We set $M_P = \lfloor \theta B \rfloor$ and $M_D = B - M_P$, and vary $\theta \in \{0.25, 0.35, 0.45, 0.55, 0.65, 0.70\}$ at $n = 250$.

Figures 7 and 8 report ALG/LP and total completion time (TCT), respectively. ALG/LP generally decreases as θ increases. By contrast, TCT rises with θ for the conversation workload and is lowest near $\theta = 0.35$ for the code workload.

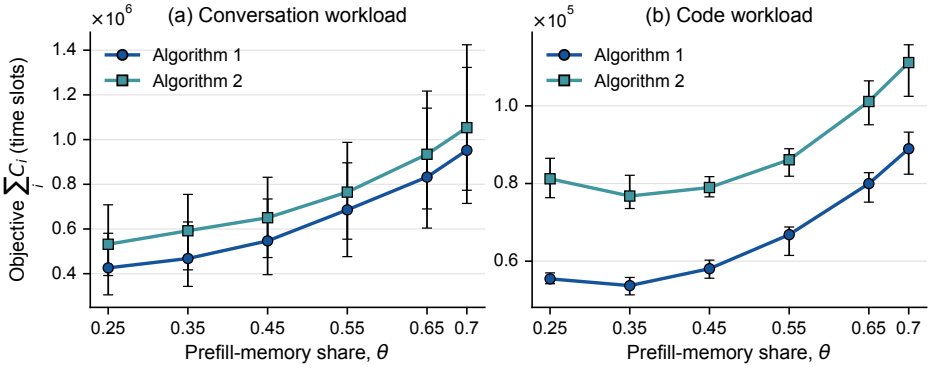


Figure 8: Total completion time versus θ for (a) conversation and (b) code workloads.

H.2 GPU Serving Results

We conduct serving experiments on two NVIDIA H100 GPUs using vLLM 0.12.0 with BF16 precision. We evaluate Qwen2.5 (7B) (Yang et al., 2024) and Llama 3.1 (8B) (Grattafiori et al., 2024) on conversation and code workloads.

For each request i, p_i is the input token count measured by the model's tokenizer. We fix the output length to d_i tokens by ignoring the end of sequence token. We set $m_i = \lceil \widehat{t}_i^P / \tau \rceil$, where $\widehat{t}_i^P$ is the prefill time estimated by interpolating measured prefill times at different prompt lengths on logarithmic scales, and τ is the measured median time per decode step for the model. For $n = 250$, $(\max_i p_i, \max_i h_i)$ equals $(4107, 4176)$ for conversation and $(7436, 7448)$ for code for both models.

P/D and Algorithms 1 and 2 use one GPU for prefill and one for decoding. Colocated runs a single tensor parallel vLLM instance across both GPUs, which execute both phases. Colocated Chunked uses the same configuration with chunked prefill enabled. Each KV cache block holds 16 tokens. For P/D and both algorithms, we configure $\lceil M_P/16 \rceil$ prefill blocks and $\lceil M_D/16 \rceil$ decode blocks. Both colocated baselines use a unified cache with $\lceil M_D/16 \rceil$ blocks.

Admission Control. An external proxy enforces the offline schedule before forwarding requests to the two vLLM instances. It waits until the full batch is registered, then applies the algorithm's group barriers and admission order. To enforce the prefill capacity, the proxy counts both requests executing prefill and requests retaining their prompt cache while awaiting decode. Before forwarding request i to decode, it checks $\sum_{j\in\mathcal{D}} h_j + h_i \le M_D$, where $\mathcal{D}$ contains requests admitted to decode but not yet completed. When request i completes, the proxy releases h_i from this accounting budget. This is a software admission rule, not a physical reservation of h_i KV cache units inside vLLM, which manages actual KV allocation separately.

KV Transfer. The prefill and decode instances transfer KV state using vLLM's P2pNcclConnector. The recorded GPU topology shows an NV12 connection between the two H100 GPUs. Although the theoretical model assumes instantaneous KV transfer, the completion times measured in the GPU experiments include KV transfer overhead.

Memory Capacity. We use $n = 250$ requests and vary $\kappa \in \{1.5, 2, 2.5, 3, 3.5, 4\}$, where $M_P = \lceil \kappa \max_i p_i \rceil$ and $M_D = \lceil \kappa \max_i h_i \rceil$.

Figure 9 shows that both algorithms reduce TCT relative to P/D across all tested capacities on the code workloads. On the conversation workloads, their TCT advantage diminishes as capacity increases.

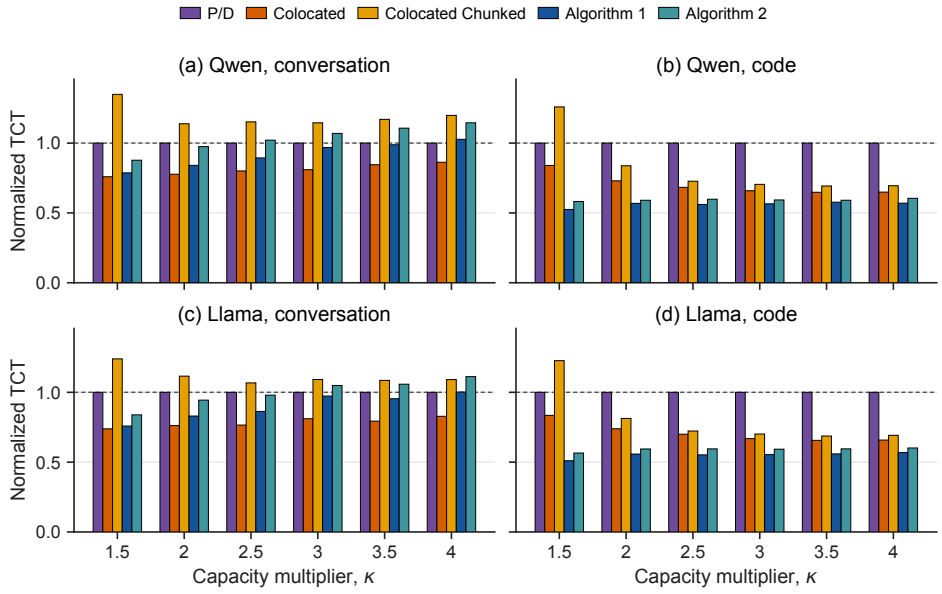


Figure 9: Total completion time as κ varies, normalized by the P/D result.

At $\kappa = 2$, Figure 10 shows that both algorithms have lower P50 but higher P99 request completion times than P/D across all four model and workload combinations. Their P95 times are lower than P/D on code workloads but higher on conversation workloads.

Prefill and Decode Demands. We use $n = 250$ requests and target three demand settings: $(\alpha, \beta) = (0.9, 0.5)$, $(0.75, 0.75)$, and $(0.5, 0.9)$, representing prefill-heavy, balanced, and decode-heavy demands, respectively.

Figure 11 shows that, for both models, both algorithms reduce TCT relative to P/D on code workloads under all three demand settings. The reductions are generally larger than those on conversation workloads. On conversation workloads, Algorithm 1 remains below P/D, whereas Algorithm 2 slightly exceeds P/D for Qwen under prefill-heavy demand.

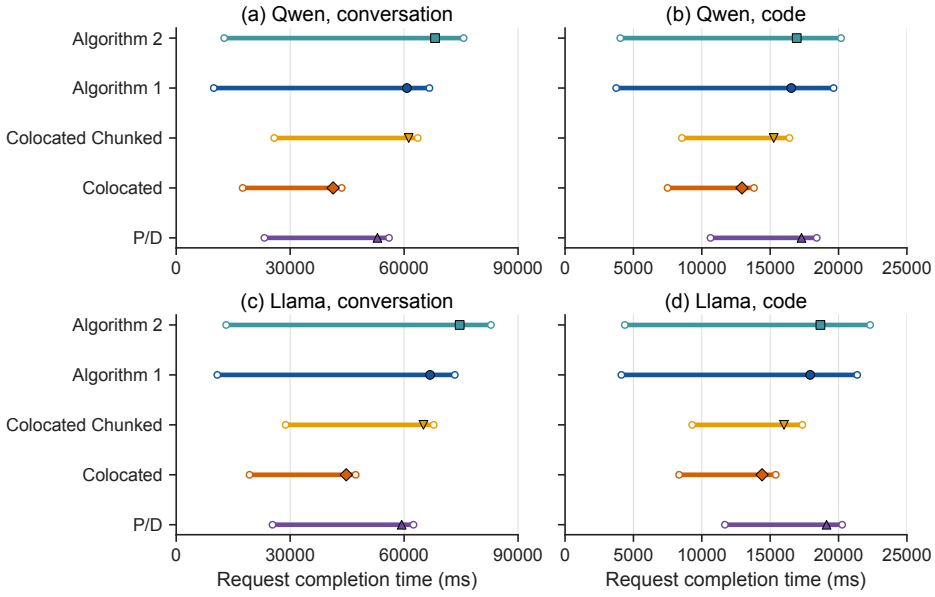


Figure 10: Request completion times at $\kappa = 2$ and $n = 250$. Each segment spans P50 to P99; the filled marker denotes P95.

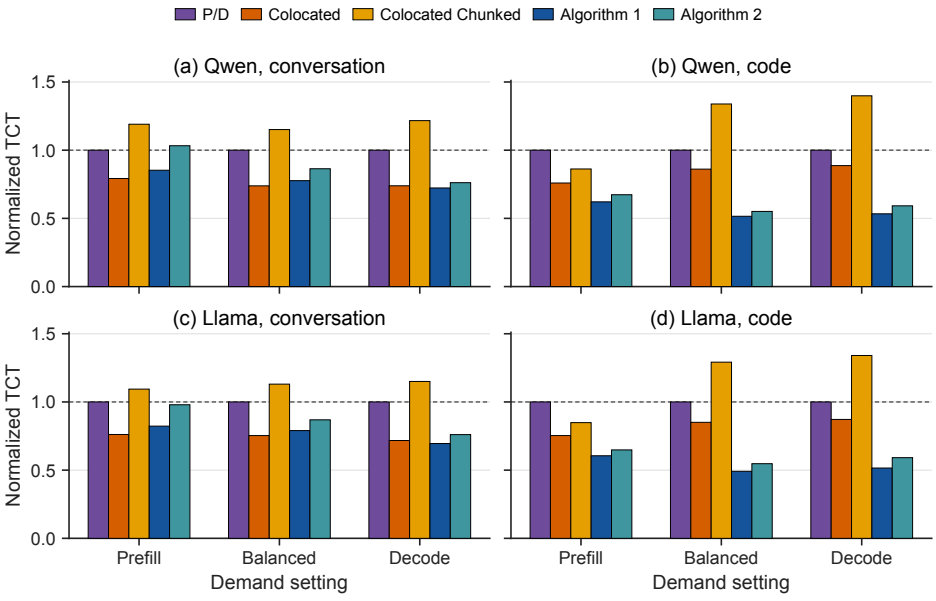


Figure 11: Total completion time for different values of α and β, normalized by the P/D result.

Time to first token (TTFT) is measured from the time each request is sent until its first output token is received. In the balanced setting, Figure 12 shows that both algorithms have lower P50 TTFT than P/D for both models and workloads. Their P99 TTFT is higher than P/D, except for Algorithm 1 on the Llama code workload, where it is slightly lower.

Number of Requests. We vary $n \in \{50, 100, 150, 200, 250\}$. The capacities are fixed across n at $(M_P, M_D) = (8214, 8352)$ for conversation and $(14872, 14896)$ for code, corresponding to the $n = 250$, $\kappa = 2$ settings.

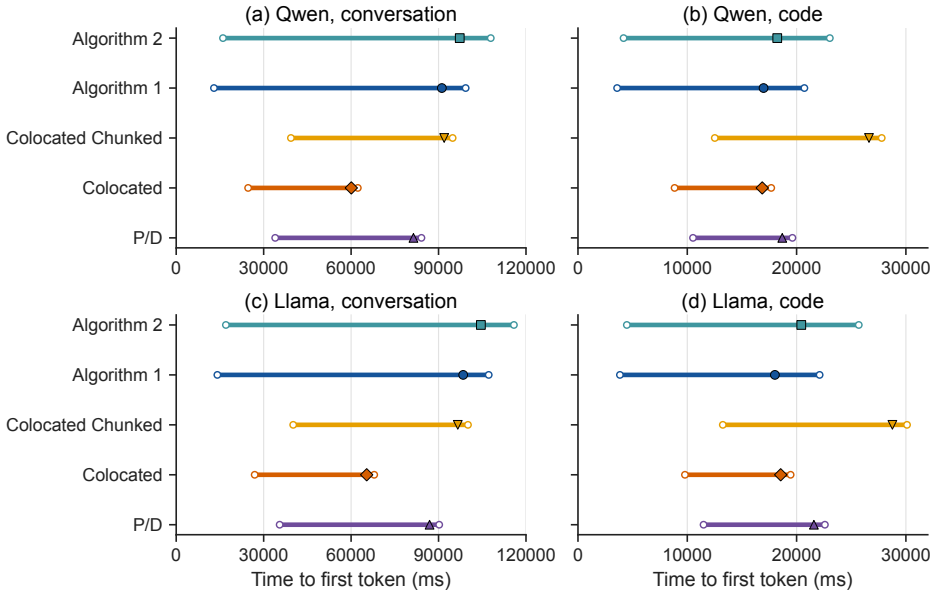


Figure 12: TTFT at $(\alpha, \beta) = (0.75, 0.75)$. Each segment spans P50 to P99; the filled marker denotes P95.

Figure 13 shows that the algorithms' normalized TCT generally decreases as n increases. Both algorithms remain below P/D across all batch sizes on code workloads; on conversation workloads, their TCT advantage appears at larger batch sizes.

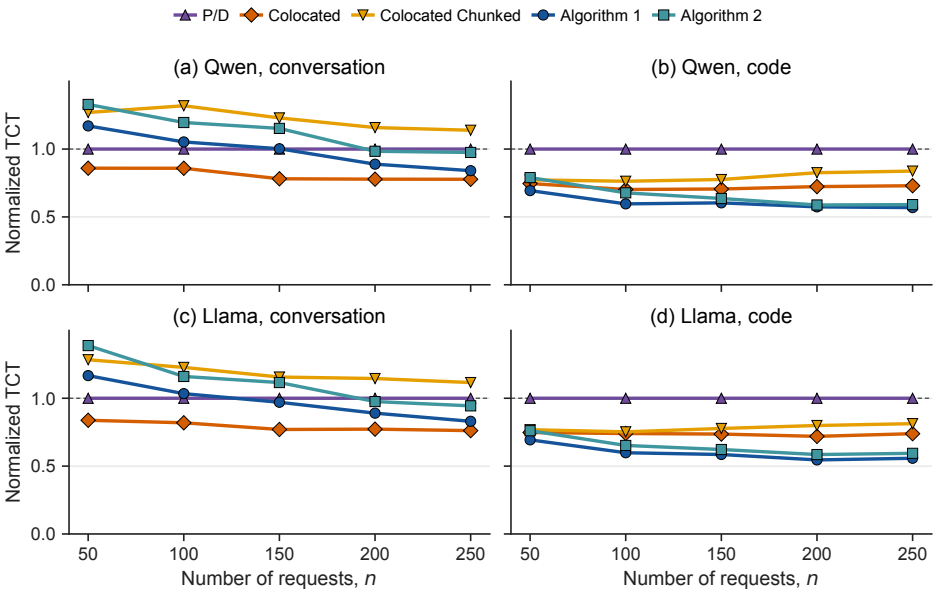


Figure 13: Total completion time as n varies, normalized by the P/D result.

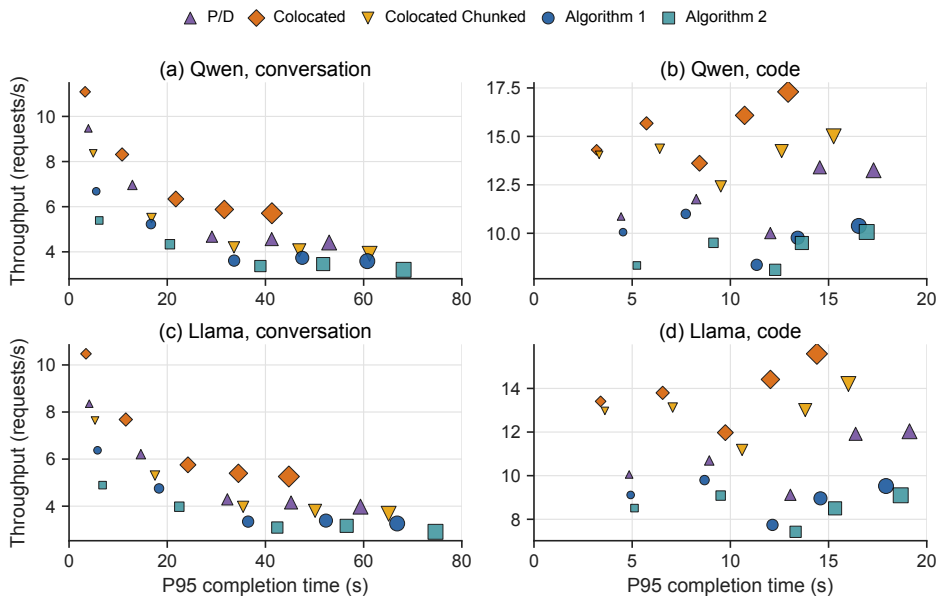


Figure 14: Throughput versus P95 request completion time as n varies. Marker size increases with n.

Figure 14 compares throughput and P95 request completion time at each batch size. At comparable values of n, Colocated achieves higher throughput and lower P95 completion time than either algorithm.

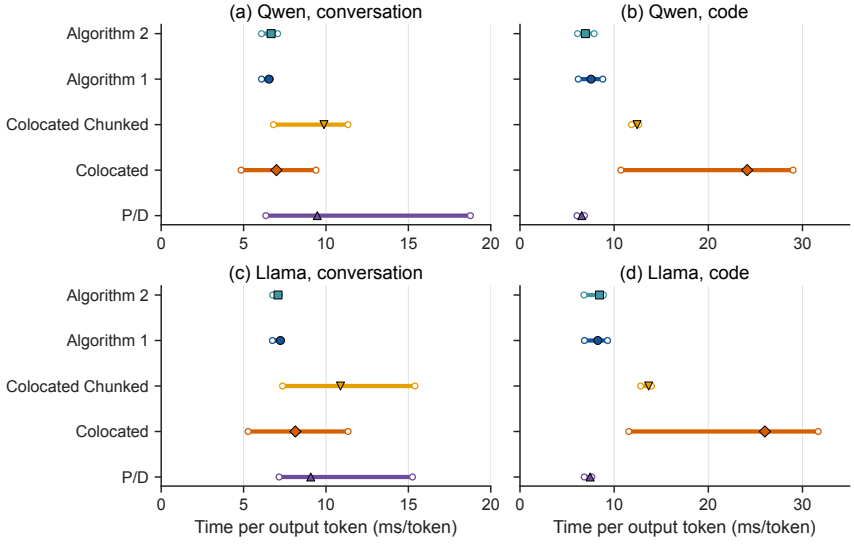


Figure 15: TPOT at $n = 250$. Each segment spans P50 to P99; the filled marker denotes P95.

Time per output token (TPOT) measures the average generation time per output token after the first token. At $n = 250$, Figure 15 shows that both algorithms have lower P95 TPOT than P/D on the conversation workloads and slightly higher P95 TPOT than P/D on the code workloads. Both algorithms remain below Colocated Chunked, while Colocated has substantially higher P95 TPOT on the code workloads.