Efficient and Robust LLM Inference Scheduling Optimization

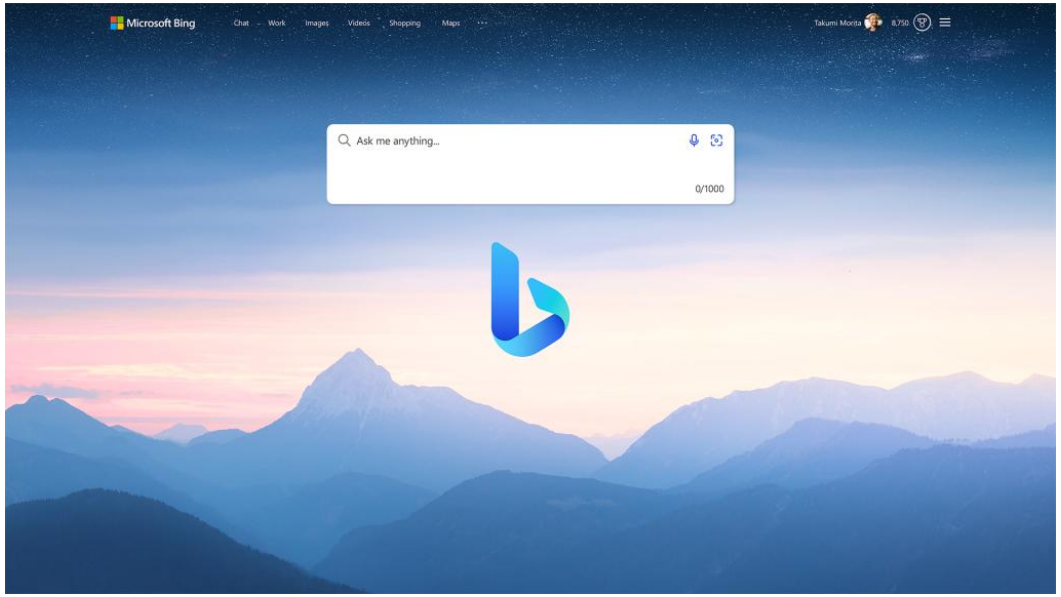
Presenter: Zijie Zhou (jerryzhou@ust.hk)
Department of Industrial Engineering & Decision Analytics, HKUST

Global Scheduling Seminar https://schedulingseminar.com/

What is Large Language Model (LLM)?

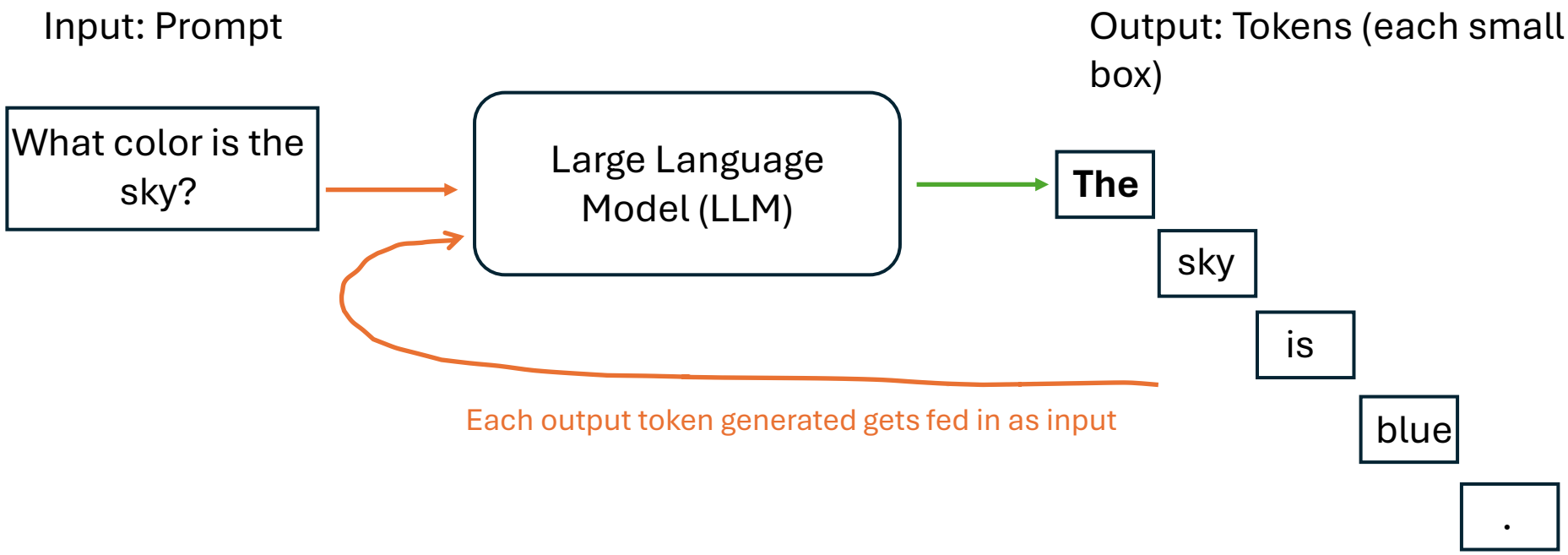
Large, general-purpose language models that can be pre-trained and then fine-tuned for specific purposes

ChatGPT



GitHub Copilot

What is LLM Inference?



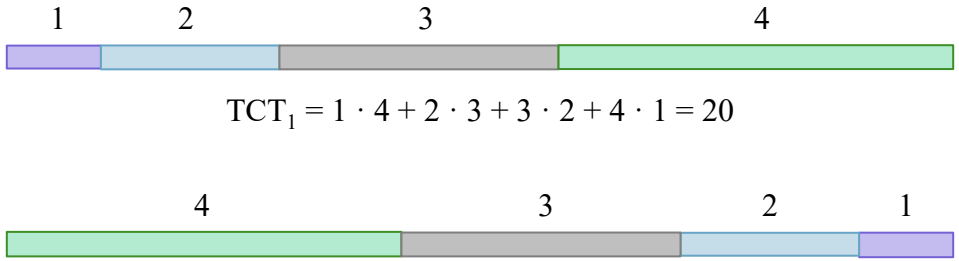
Research Background

A core metric for evaluating large language model (LLM) efficiency is the **Total Completion Time (TCT)**—the sum of each request's completion time (from a unified start at t=0). Unlike makespan (total duration to finish all requests), TCT penalizes individual request delays, making it critical for user-facing latency.

This metric hinges on three factors:
- Request characteristics: Prompt size and output length are predetermined by user inputs.
- Processing Speed: Accelerating execution risks degrading output quality—an unacceptable tradeoff for reliable LLMs.
- **Scheduling Algorithms:** This is the only adjustable lever to minimize TCT by reordering requests without hardware changes or accuracy loss.

Therefore, our goal is to design a scheduling algorithm that dynamically optimize TCT by prioritizing requests based on prompt-output patterns and system constraints.

Example:



$TCT_1 = 1 \cdot 4 + 2 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 = 20$

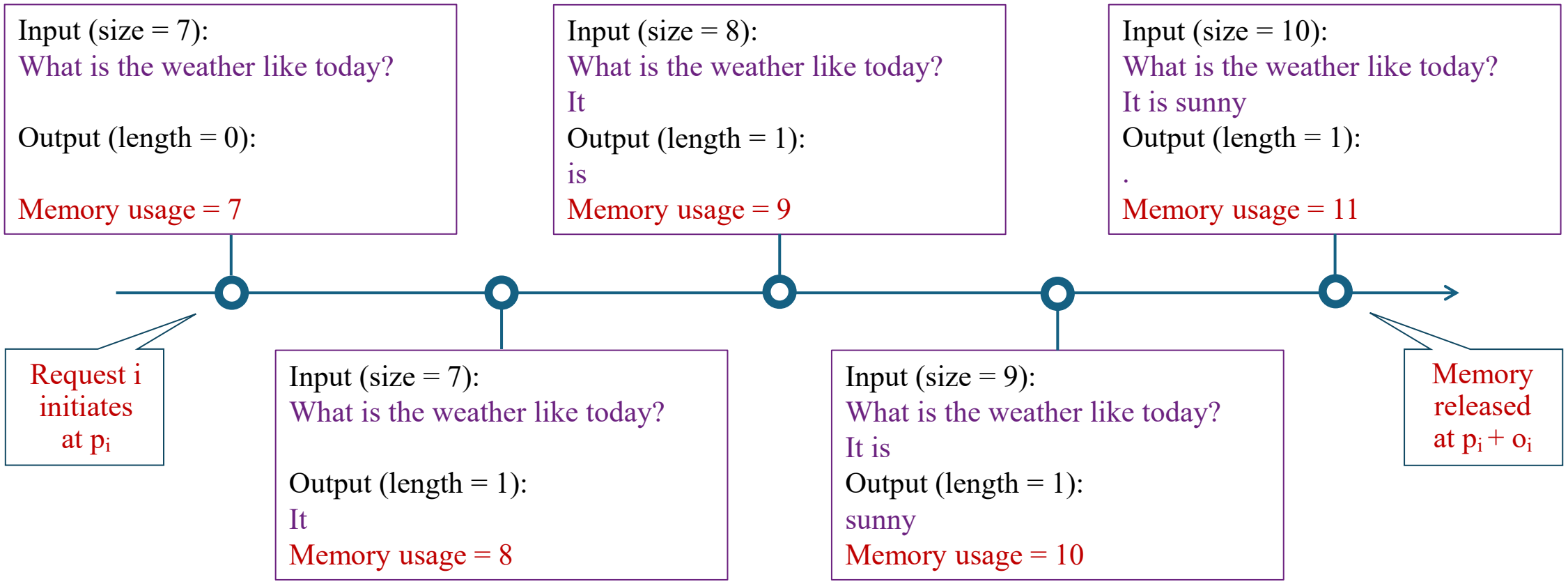
$TCT_2 = 4 \cdot 4 + 3 \cdot 3 + 2 \cdot 2 + 1 \cdot 1 = 30$

Same makespan, but different total completion time.

LLM Inference for One Request

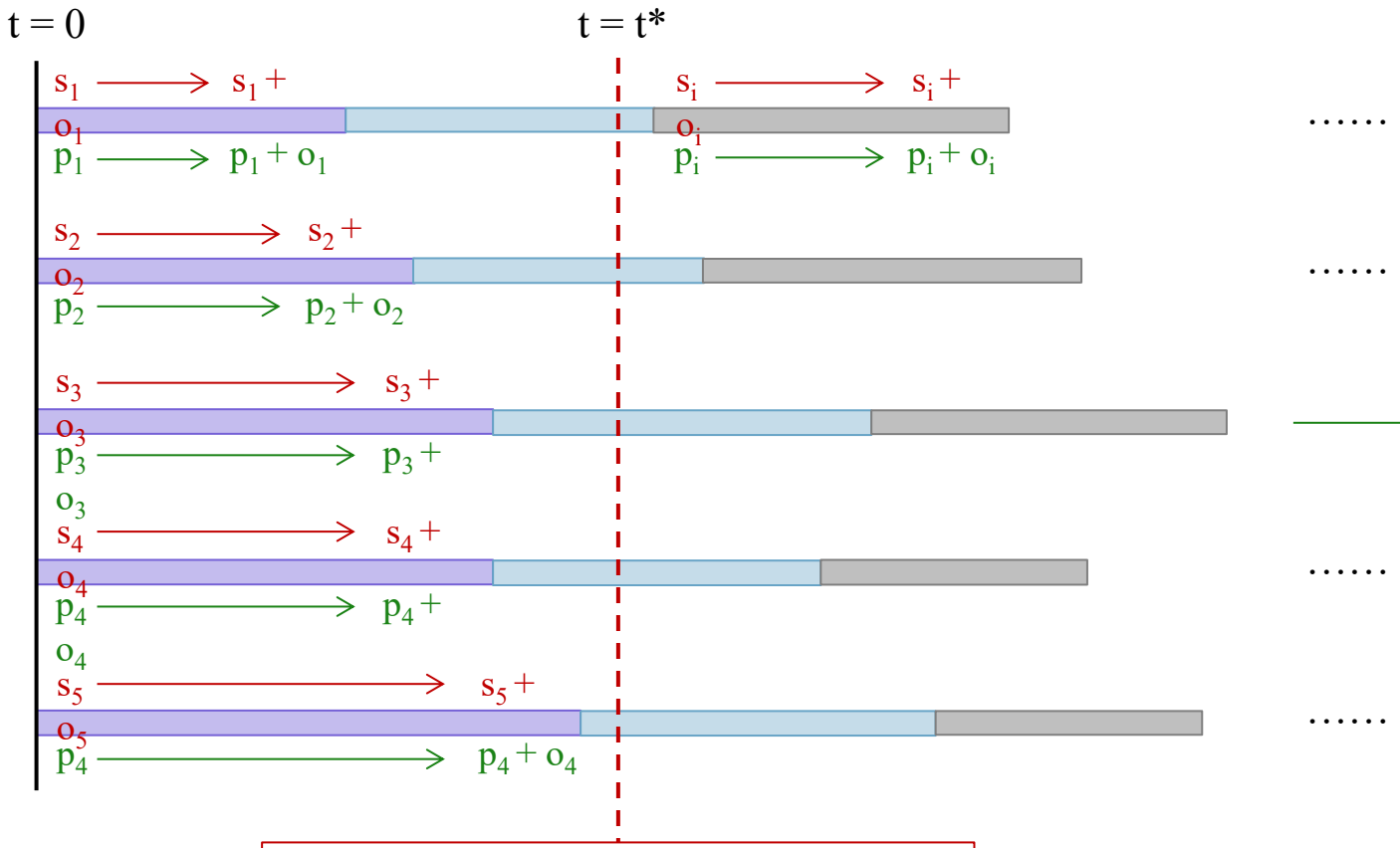
Request r_i:
- Prompt (size $s_i = 7$): What is the weather like today?
- Response (length $o_i = 4$): It is sunny.
- Initiating time: p_i

Remarks:
- Each output token generated get fed in as input
- At time t, memory usage = $s_i + t - p_i$



LLM Inference for Multiple Requests

Request r_i -- Prompt size s_i -- Response length o_i -- Initiating time p_i



Memory constraints:
At any time t*, the total memory usage must not exceed the memory limit M.

Performance Metric:

Let ALG denote our algorithm and OPT denote the optimal schedule. Given any list of requests $\mathcal{I}$, let $\mathrm{TCT}_{\mathcal{I}}(\Lambda)$ denote the total completion time of schedule Λ:

$$\mathrm{TCT}_{\mathcal{I}}(\Lambda) = \sum_{r_i \in \mathcal{I}} (p_i(\Lambda) + o_i),$$

where $p_i(\Lambda)$ denotes the starting time to process r_i under schedule Λ. The optimal schedule is defined as

$$\mathrm{OPT} = \arg\min_{\Lambda} \mathrm{TCT}_{\mathcal{I}}(\Lambda).$$

The competitive ratio is given by

$$\mathrm{CR}(\mathrm{ALG}) = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})}.$$

Literature Review

(Jaillet et al. 2025) first suggests a model which captures the KV cache memory constraint in LLM inference, and designs an efficient scheduling algorithm based on “Shortest-Job-First”. However, there are two key assumptions:

1. All input sizes are homogeneous.

2. All output length are perfectly predictable.

Under assumption 1, “Shortest-Job-First” makes sense because:

- Shorter output jobs process faster (need fewer batches to process).

- Shorter output jobs need smaller amount of memory, which inproves the concurrency at the beginning.

LLM Serving Optimization with Variable Prefill and Decode Lengths

Meixuan Wang (wangmx22@mails.tsinghua.edu.cn)
Department of Computer Science and Technology, Tsinghua University
Yinyu Ye (yyye@stanford.edu)
Departmemt of Management Science and Engineering, Stanford
Department of Industrial Engineering & Decision Analytics, HKUST
Zijie Zhou (jerryzhou@ust.hk)
Department of Industrial Engineering & Decision Analytics, HKUST

NP-Hard Statements

THEOREM 1. *Minimizing the makespan is NP-hard.*

Proof of Theorem 1 We prove NP-hardness via a reduction from the Partition problem. Consider the following Partition Problem: Given a multiset of integers $X = \{x_1, \ldots, x_n\}$ summing to $2T$, does there exist a partition of X into two subsets S_1, S_2 such that $\sum_{i \in S_1} x_i = \sum_{i \in S_2} x_i = T$?

We construct the following reduction:

- For each integer x_i, create a request i with $s_i = x_i$ and $o_i = 1$
- Set memory capacity $M = T$.

We show that the Partition instance has a solution if and only if the constructed instance admits a schedule with makespan 2.

Case 1: Partition Exists. Given subsets S_1, S_2 each summing to T, the schedule:

- Batch 1: Process all requests in S_1 (memory usage $= T$)
- Batch 2: Process all requests in S_2 (memory usage $= T$)

completes all requests in makespan 2.

Case 2: No Partition Exists. Any valid schedule must use at least 3 batches because:

- No single batch can process all requests (total memory $2T > M$)
- Any two-batch solution would require both batches to have memory exactly T, which would constitute a valid partition

Thus, the makespan is at least 3. The makespan equals 2 if and only if the Partition instance has a solution, proving NP-hardness of makespan minimization. □

THEOREM 2. *Minimizing the total completion time is NP-hard.*

Proof of Theorem 2 We prove this via a reduction from the 3-Partition problem, which is known to be strongly NP-hard. Consider the following 3-Partition Problem: Given a multiset of integers $X = \{x_1, \ldots, x_{3m}\}$ summing to mT, where each x_i satisfies $T/4 < x_i < T/2$, the problem asks whether X can be partitioned into m disjoint subsets $S_1, \ldots, S_m$ such that the sum of each subset equals T.

Then, we construct the following reduction: Given an instance of 3-Partition, we construct an instance as follows:

- For each integer x_i, create a request i with $s_i = x_i$ and $o_i = 1$.
- Set the memory capacity $M = T$.

We show that the 3-Partition instance has a solution if and only if the constructed instance has a schedule with total completion time $\text{TCT} = \frac{3m(m+1)}{2}$.

Case 1: 3-Partition Exists. Suppose X can be partitioned into m subsets $S_1, \ldots, S_m$, each summing to T. Then, consider the schedule which processes batch k with all requests in S_k. The schedule is feasible since the memory usage of each batch is M. Moreover, as each batch processes exactly 3 requests (due to $T/4 < x_i < T/2$), the total completion time is:

$$\text{TCT} = \sum_{t=1}^{m} 3t = \frac{3m(m+1)}{2}.$$

Case 2: No 3-Partition Exists. If no such partition exists, then at least one batch must process fewer than 3 requests (since no subset of 2 requests sums to $\leq T$, given $x_i > T/4$). This forces the schedule to use at least $m+1$ batches. Compared to the schedule in case 1, there is at least one job having to be swapped from one of the batch 1 to batch m to the batch $m+1$, and the total completion time in this case is strictly greater than $\frac{3m(m+1)}{2}$.

Since the minimal total completion time is $\frac{3m(m+1)}{2}$ if and only if the 3-Partition instance has a solution, the problem of minimizing total completion time is NP-hard. □

Baseline Algorithms

First-Come-First-Serve (FCFS):

- Processes requests in arrival order.

Performance Metric:

The competitive ratio is given by

$$\mathrm{CR}(\mathrm{ALG}) = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})}.$$

Shortest-First (SF):

- Prioritizes requests with the smallest response length (o_i) first.
- Is a special case of FCFS when shorter requests arrive earlier.

Observations:

- When prompt size (s_i) are the same, **SF** is a very good algorithm.
- However, when prompt size (s_i) is variable, neither **FCFS** nor **SF** achieves a constant competitive ratio (CR).

Intuition:

- Requests with a very large prompt size (o_i), despite having a small response length (s_i), drastically limit the amount of requests processed early, significantly increasing the total completion time (TCT).

Example: (Memory limit = M)

Type	s_i	o_i	Amount
1	$M^{0.5} - 1$	1	M
2	1	2	$M^{1.5}$

Under **SF**, type 1 requests are prioritized:

$$\mathrm{TCT}_{\mathcal{I}}(\mathrm{SF}) = \sqrt{M} \sum_{i=1}^{X/\sqrt{M}} i + \frac{XY}{\sqrt{M}} + \frac{2M}{3} \sum_{j=1}^{3Y/M} j.$$

Consider **A** that prioritizes type 2 requests:

$$\mathrm{TCT}_{\mathcal{I}}(\mathrm{A}) = \frac{2M}{3} \sum_{j=1}^{3Y/M} j + \frac{6XY}{M} + \sqrt{M} \sum_{i=1}^{X/\sqrt{M}} i.$$

Compare the two algorithms:

$$\frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{SF})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{A})} = \frac{\frac{1}{2}M^{1.5} + 3M^{1.25} + M^2}{\frac{1}{2}M^{1.5} + 3M^{1.25} + 6M^{1.5}} \geq \frac{2}{13}M^{0.5}.$$

Quality Metric

Inspired by the shortcomings of **Shortest-First (SF)** , we introduce a novel quality metric to determine request prioritization. For any list of requests $\mathcal{X}$,

$$F(\mathcal{X}) = \frac{\sum_{r_i \in \mathcal{X}} o_i}{|\mathcal{X}|^2},$$

where a smaller value indicates higher priority.

This metric jointly optimizes low average response lengths and high batch throughput. Unlike **SF**—which prioritizes requests with small s_i regardless of o_i—our metric deprioritizes requests with small s_i but excessively large o_i, thus preventing inefficient early-stage scheduling.

Example: (Memory limit = M)

Consider a simple scenario with two batches, where requests are homogeneous within each batch but heterogeneous across batches. We want to determine which batch to prioritize to minimize the total completion time (TCT).

Batch	s_i	o_i	Amount	Metric
1	s_1	o_1	n_1	o_1/n_1
2	s_2	o_2	n_2	o_2/n_2

If we prioritize batch 1 over batch 2, the TCT is $o_1n_1+o_2n_2+o_1n_2$. Conversely, if we prioritize batch 2 over batch 1, the TCT becomes $o_1n_1+o_2n_2+o_2n_1$. Comparing the metrics, if $o_1/n_1 < o_2/n_2$, then $o_1n_2 < o_2n_1$, so batch 1 should be prioritized. Otherwise, batch 2 should be prioritized. Therefore, this decision rule aligns perfectly with our quality metric's selection criteria.

Our Algorithm

Based on the quality metric

$$F(\mathcal{X}) = \frac{\sum_{r_i \in \mathcal{X}} o_i}{|\mathcal{X}|^2},$$

we sort requests by iteratively selecting from the unsorted pool the sublist $\mathcal{X}$ that minimizes $F(\mathcal{X})$ while respecting memory constraints, then ordering requests in $\mathcal{X}$ by ascending o_i:

Input: $\mathcal{I}$, **Output**: $\mathcal{I}'$

$\mathcal{I}' \leftarrow []$

while $\mathcal{I}$ **do**

  $\mathcal{X} \leftarrow \arg\min_{\mathcal{X} \subseteq \mathcal{I}}(F(\mathcal{X}), -|\mathcal{X}|)$ **subject to** $M(\mathcal{X}, p_i + o_i) \leq M, \forall r_i \in \mathcal{X}$

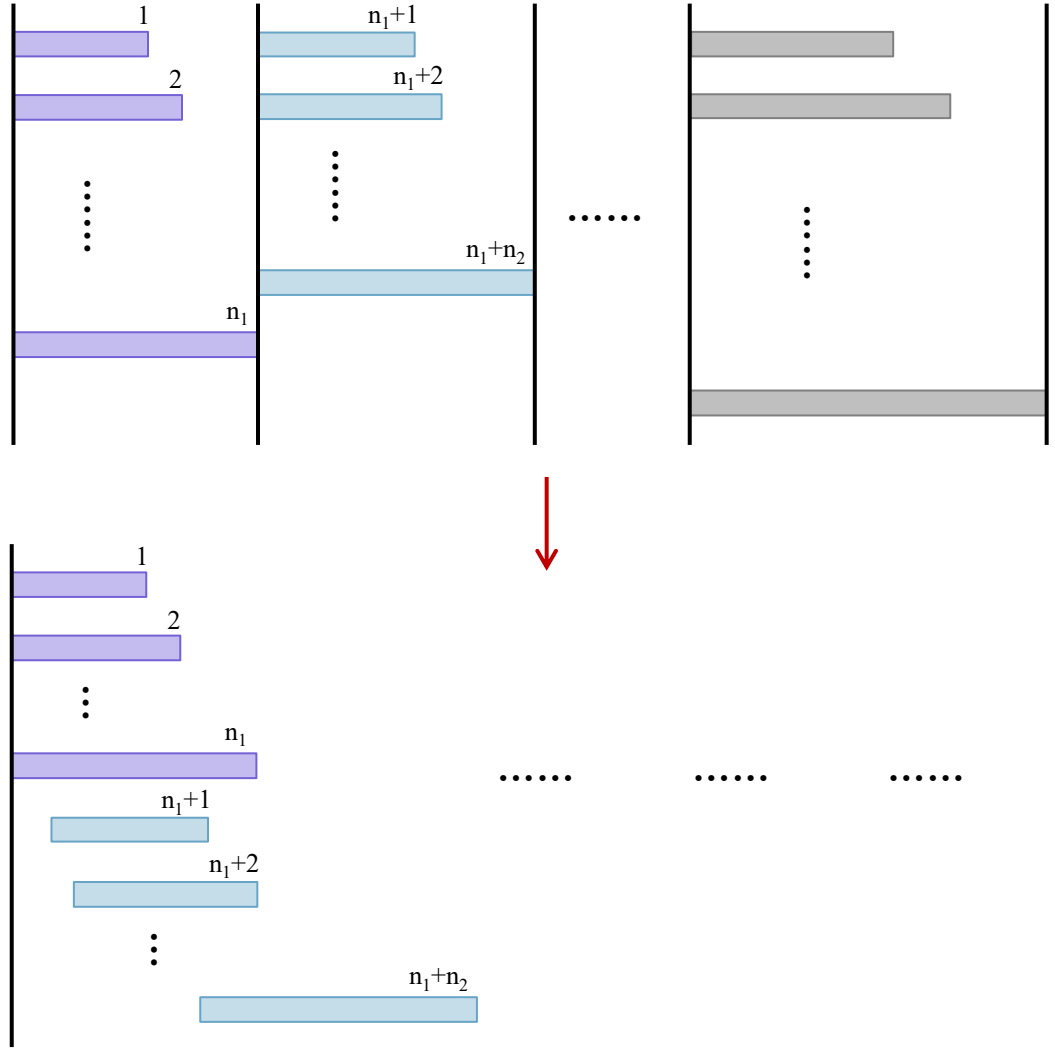
  Sort $\mathcal{X}$ in ascending order of o_i

  $\mathcal{I}' \leftarrow \mathcal{I}' + \mathcal{X}, \mathcal{I} \leftarrow \mathcal{I} - \mathcal{X}$

end while

Following this order, we schedule requests sequentially at their earliest feasible times while respecting memory constraints.

Sketch:



Constant-CR Proof: Step 1, ALG → $ALG_{separate}$

We denote our algorithm as **ALG** and the optimal schedule as **OPT**, with corresponding total completion times TCT(ALG) and TCT(OPT). We establish the constant competitive ratio through the following inequality chain:

$$TCT(ALG) \le TCT(ALG_{separate}) \le 4 \cdot TCT(ALG_{group}) \le 8 \cdot TCT(ALG_{align}) \le 8 \cdot TCT(OPT_{transform}) \le 48 \cdot TCT(OPT).$$

Our algorithm ALG operates in two phases: first, it sorts all requests based on the quality metric; then, it sequentially schedules each request at the earliest available time slot. To establish an upper bound for TCT(ALG), we consider an alternative approach where we reverse the second phase by handling complete batches sequentially one at a time, as illustrated on the right. This modified scheduling scheme, which we denote as $ALG_{separate}$, satisfy the inequality:

$$TCT(ALG) \le TCT(ALG_{separate}).$$

Performance Metric:

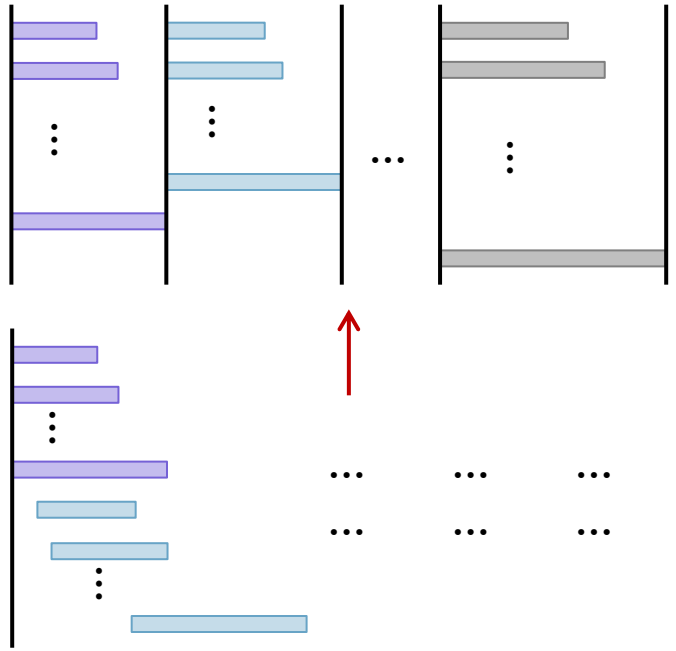
The competitive ratio is given by

$$CR(ALG) = \sup_{\mathcal{I}} \frac{TCT_{\mathcal{I}}(ALG)}{TCT_{\mathcal{I}}(OPT)}.$$

Assumption:

s_i, o_i are small quantities relative to M.

Sketch:



Constant-CR Proof: Step 2, $\text{ALG}_{\text{separate}} \rightarrow \text{ALG}_{\text{group}}$

Let the requests in $\mathcal{X}_k$, ordered by increasing response length, be denoted as $r_{k+1,1}, r_{k+1,2}, \ldots, r_{k+1,n_k}$, where $n_k = |\mathcal{X}_k|$. Since $M \to \infty$ and $s_i, o_i = \epsilon(M)$, $n_k \to \infty$.

We aggregate the batches $\{\mathcal{X}_k\}$, generated by Algorithm 1, into larger groups $\{\mathcal{Y}_m\}$ constructed as

$$\mathcal{Y}_m = \mathcal{X}_{b_{m-1}+1} + \mathcal{X}_{b_{m-1}+2} + \cdots + \mathcal{X}_{b_m}$$

where $b_m = \sum_{j=1}^{m} a_j$, a_j denotes the number of batches in $\mathcal{Y}_j$, and $\mathcal{A}_1 + \mathcal{A}_2 + \cdots + \mathcal{A}_n$ represents the merged batch formed by processing all requests from $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ concurrently. Each $\mathcal{Y}_m$ is designed so that $\mathcal{X}_{b_m} + [r_{b_m+1,1}]$ exceeds memory M. Specifically, for any $\mathcal{X}_{b_m}$ and $r_{b_m+1,1}$, there exists a time

$$t^* \in \{p_i + o_i \,|\, r_i \in \mathcal{X}_{b_m} + [r_{b_m+1,1}]\}$$

such that

$$M(\mathcal{X}_{b_m} + [r_{b_m+1,1}], t^*) > M.$$

When this condition holds, we say $\mathcal{X}_{b_m}$ nearly saturates memory M, which implies

$$\sum_{r_i \in \mathcal{X}_{b_m}} (s_i + o_i) > M - \epsilon(M).$$

To achieve this, we consider the following two cases.

Performance Metric:

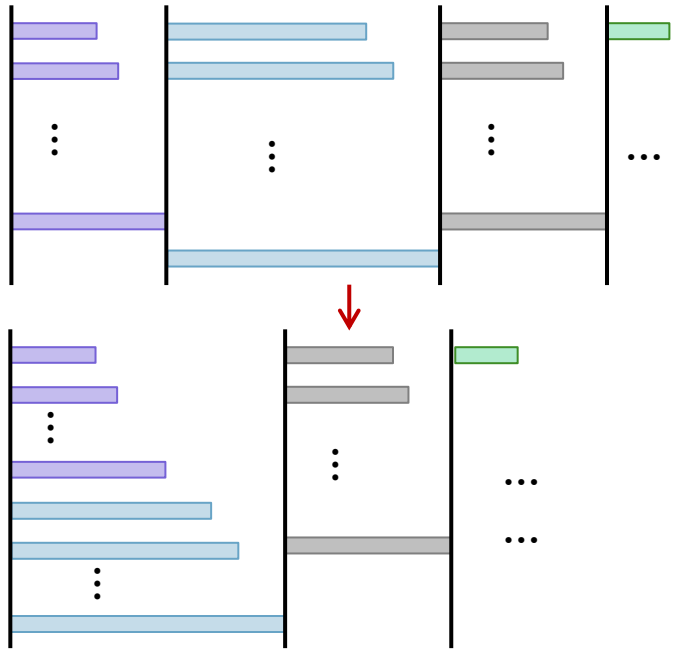
The competitive ratio is given by

$$\text{CR}(\text{ALG}) = \sup_{\mathcal{I}} \frac{\text{TCT}_{\mathcal{I}}(\text{ALG})}{\text{TCT}_{\mathcal{I}}(\text{OPT})}.$$

Assumption:

s_i, o_i are small quantities relative to M.

Sketch:



Constant-CR Proof: Step 2, $ALG_{separate} \rightarrow ALG_{group}$

Case 1: The request highlighted with a red circle in the sketch corresponds to this case.

Assume $\mathcal{X}_1, \mathcal{X}_2, \cdots, \mathcal{X}_k$ have already been grouped. For the consecutive batches $\mathcal{X}_{k+1}$ and $\mathcal{X}_{k+2}$, we consider two cases. First, if

$$o_{k+2,1} \leq \frac{2n_{k+1}+1}{n_{k+1}^2} \cdot \sum_{j=1}^{n_{k+1}} o_{k+1,j},$$

then $\mathcal{X}_{k+1} + [r_{k+2,1}]$ exceeds memory, because otherwise, $r_{k+2,1}$ should have been included in $\mathcal{X}_{k+2}$ to either reduce $F(\mathcal{X}_{k+2})$ or increase $|\mathcal{X}_{k+2}|$ while maintaining the same value of $F(\mathcal{X}_{k+2})$. This, however, contradicts the selection criterion of Algorithm 1. Therefore, $\mathcal{X}_{k+1}$ nearly saturates memory M and can consequently be treated as an independent group.

Performance Metric:

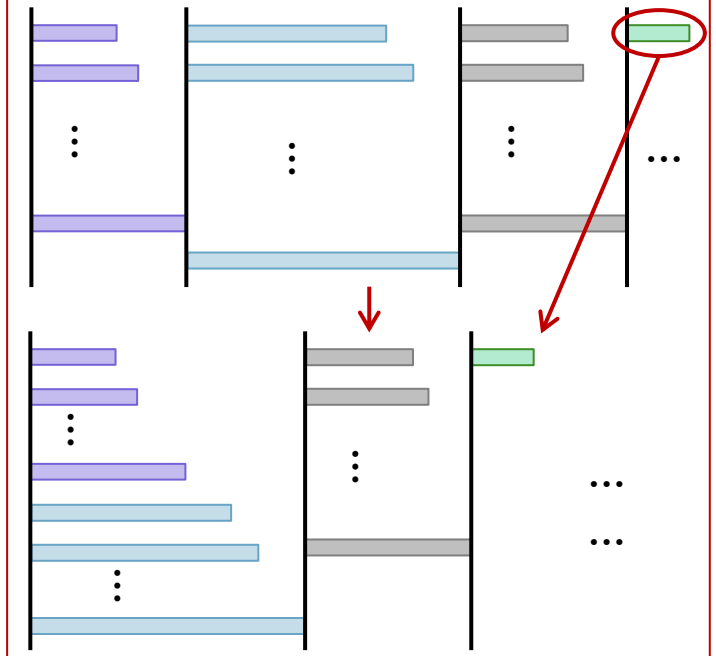
The competitive ratio is given by

$$\mathrm{CR}(\mathrm{ALG}) = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})}.$$

Assumption:

s_i, o_i are small quantities relative to M.

Sketch:



Constant-CR Proof: Step 2, $\mathrm{ALG}_{\mathrm{separate}} \rightarrow \mathrm{ALG}_{\mathrm{group}}$

Case 2: The request highlighted with a red circle in the sketch corresponds to this case.

Alternatively, if

$$o_{k+2,1} > \frac{2n_{k+1}+1}{n_{k+1}^2} \cdot \sum_{j=1}^{n_{k+1}} o_{k+1,j},$$

then $\mathcal{X}_{k+1} + [r_{k+2,1}]$ does not necessarily exceed the memory limit M.

We iteratively merge consecutive batches until encountering a batch that satisfies **Case 1**. For all merged batches, we establish bounding inequalities on their makespans—with the quality metric serving as the critical determinant:

$$\begin{aligned} o_{k+2,n_{k+2}} &\geq \frac{\sum_{j=1}^{q_{k+2}} o_{k+2,j}}{q_{k+2}} \\ &> \left(2+\frac{q_{k+2}}{n_{k+1}}\right) \cdot \frac{\sum_{i=1}^{n_{k+1}} o_{k+1,i}}{n_{k+1}} \\ &\geq \left(2+\frac{q_{k+2}}{n_{k+1}}\right) \cdot \frac{n_{k+1}^2 - 2n_{k+1}+1}{2n_{k+1}^2 - n_{k+1}} \cdot o_{k+1,n_{k+1}} \\ &> 2 \cdot \frac{1}{2} \cdot o_{k+1,n_{k+1}} \\ &= o_{k+1,n_{k+1}}. \end{aligned}$$

$$\begin{aligned} o_{k+3,n_{k+3}} &\geq \frac{\sum_{l=1}^{q_{k+3}} o_{k+3,l}}{q_{k+3}} \\ &> \left(2+\frac{q_{k+3}}{n_{k+2}}\right) \cdot \frac{\sum_{i=1}^{n_{k+2}} o_{k+2,i}}{n_{k+2}} \\ &\geq \left(2+\frac{q_{k+3}}{n_{k+2}}\right) \cdot \frac{\sum_{j=1}^{q_{k+2}} o_{k+2,j}}{q_{k+2}} \\ &\geq \left(2+\frac{q_{k+3}}{n_{k+2}}\right) \cdot \left(2+\frac{q_{k+2}}{n_{k+1}}\right) \cdot \frac{\sum_{i=1}^{n_{k+1}} o_{k+1,i}}{n_{k+1}} \\ &= 2 \cdot o_{k+1,n_{k+1}}. \end{aligned}$$

Similarly,

$$o_{k+i,n_{k+i}} \leq \left(\frac{1}{2}\right)^{\lfloor \frac{a-i}{2} \rfloor} \cdot o_{k+a,n_{k+a}}.$$

Performance Metric:

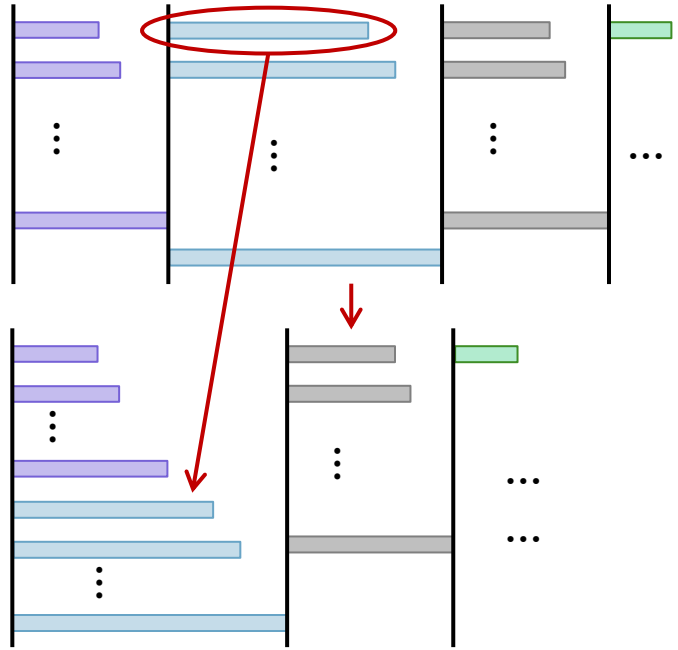
The competitive ratio is given by

$$\mathrm{CR}(\mathrm{ALG}) = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})}.$$

Assumption:

s_i, o_i are small quantities relative to M.

Sketch:



Constant-CR Proof: Step 2, $ALG_{separate} \rightarrow ALG_{group}$

Subsequently, we obtain

$$\sum_{i=1}^{a} o_{k+i,n_{k+i}} \le \sum_{i=1}^{a} \left(\frac{1}{2}\right)^{\lfloor \frac{a-i}{2} \rfloor} \cdot o_{k+a,n_{k+a}}$$

$$\le 2 \cdot \frac{1-\left(\frac{1}{2}\right)^{\lfloor \frac{a-1}{2} \rfloor}}{1-\frac{1}{2}} \cdot o_{k+a,n_{k+a}}$$

$$\le 4 \cdot o_{k+a,n_{k+a}}.$$

Following this procedure, we begin by forming the first batch group as $\mathcal{Y}_1 = \mathcal{X}_1 + \mathcal{X}_2 + \cdots + \mathcal{X}_{a_1}$. Once $\mathcal{Y}_m$ is constructed, we proceed to form the next group as

$$\mathcal{Y}_{m+1} = \mathcal{X}_{b_m+1} + \mathcal{X}_{b_m+2} + \cdots + \mathcal{X}_{b_{m+1}}.$$

By this approach, we effectively combine $\{\mathcal{X}_k\}$ to form larger groups $\{\mathcal{Y}_m\}$.

Through rigorous derivation (omitted for brievity), we prove that the modified scheduling scheme $\mathbf{ALG_{group}}$ satisfies the following performance guarantee:

$$\mathbf{TCT(ALG_{separate}) < 4 \cdot TCT(ALG_{group}).}$$

Performance Metric:

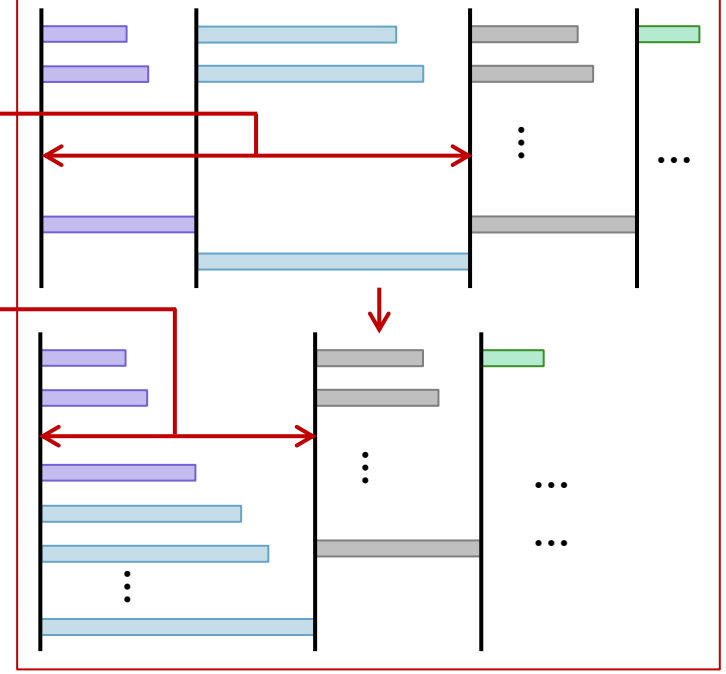
The competitive ratio is given by

$$\mathrm{CR(ALG)} = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})}.$$

Assumption:

s_i, o_i are small quantities relative to M.

Sketch:



Constant-CR Proof: Step 3, $\mathrm{ALG}_{\mathrm{group}} \to \mathrm{ALG}_{\mathrm{align}}$

We now focus on the terminal batches $\{\mathcal{X}_{b_m}\}$. For further analysis, we replace each request's original response length in $\mathcal{X}_{b_m}$ with the batch's average response length:

$$\bar{o}_{b_m} = \frac{\sum_{i=1}^{n_{b_m}} o_{b_m,i}}{n_{b_m}}.$$

Let $\mathrm{ALG}_{\mathrm{align}}$ denote the new schedule after this replacement. As proved before,

$$\bar{o}_{b_m} \geq \frac{1}{2} \cdot o_{b_m,n_{b_m}}. \tag{14}$$

Given any list of requests $\mathcal{I}$, by inequality (14), we derive

$$\begin{aligned}
\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG}_{\mathrm{align}}) &= \sum_{m=1}^{\infty}\left(\bar{o}_{b_m} \cdot \sum_{k=b_m+1}^{\infty} n_k\right) + \sum_{k=1}^{\infty}\sum_{i=1}^{n_k} o_{k,i} \\
&\geq \sum_{m=1}^{\infty}\left(\frac{1}{2} \cdot o_{b_m,n_{b_m}} \cdot \sum_{k=b_m+1}^{\infty} n_k\right) + \sum_{k=1}^{\infty}\sum_{i=1}^{n_k} o_{k,i} \\
&\geq \frac{1}{2} \cdot \left(\sum_{m=1}^{\infty}\left(o_{b_m,n_{b_m}} \cdot \sum_{k=b_m+1}^{\infty} n_k\right) + \sum_{k=1}^{\infty}\sum_{i=1}^{n_k} o_{k,i}\right) \\
&= \frac{1}{2} \cdot \mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG}_{\mathrm{group}}).
\end{aligned} \tag{15}$$

So far, we have completed the first half of the proof, which gives

$\mathrm{TCT}(\mathrm{ALG}) \leq \mathrm{TCT}(\mathrm{ALG}_{\mathrm{separate}})$

$\leq 4 \cdot \mathrm{TCT}(\mathrm{ALG}_{\mathrm{group}}) \leq 8 \cdot \mathrm{TCT}(\mathrm{ALG}_{\mathrm{align}})$.

Additionally, since $o_{k+1,n_{k+1}} \in \mathcal{X}_{k+1}$, it follows that

$$\frac{\sum_{i=1}^{n_{k+1}-1} o_{k+1,i}}{(n_{k+1}-1)^2} \geq \frac{\sum_{i=1}^{n_{k+1}} o_{k+1,i}}{n_{k+1}^2},$$

which implies the inequality

$$\frac{\sum_{i=1}^{n_{k+1}} o_{k+1,i}}{n_{k+1}} \geq \frac{n_{k+1}^2 - 2n_{k+1} + 1}{2n_{k+1}^2 - n_{k+1}} \cdot o_{k+1,n_{k+1}} = \frac{1}{2} \cdot o_{k+1,n_{k+1}}, \tag{5}$$

Performance Metric:

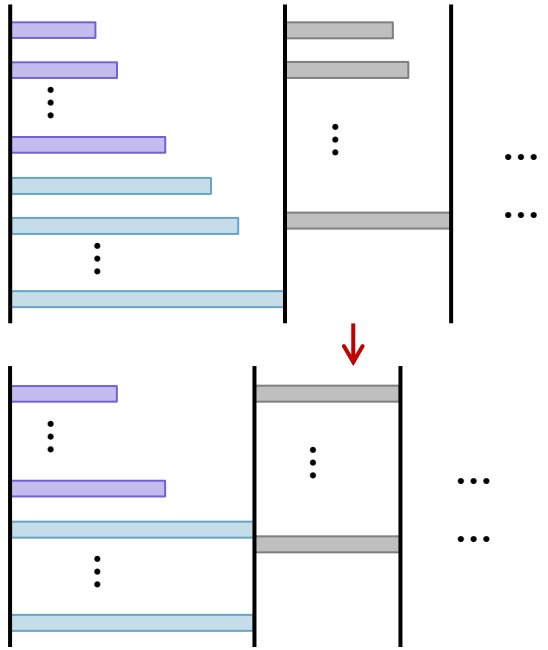
The competitive ratio is given by

$$\mathrm{CR}(\mathrm{ALG}) = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})}.$$

Assumption:

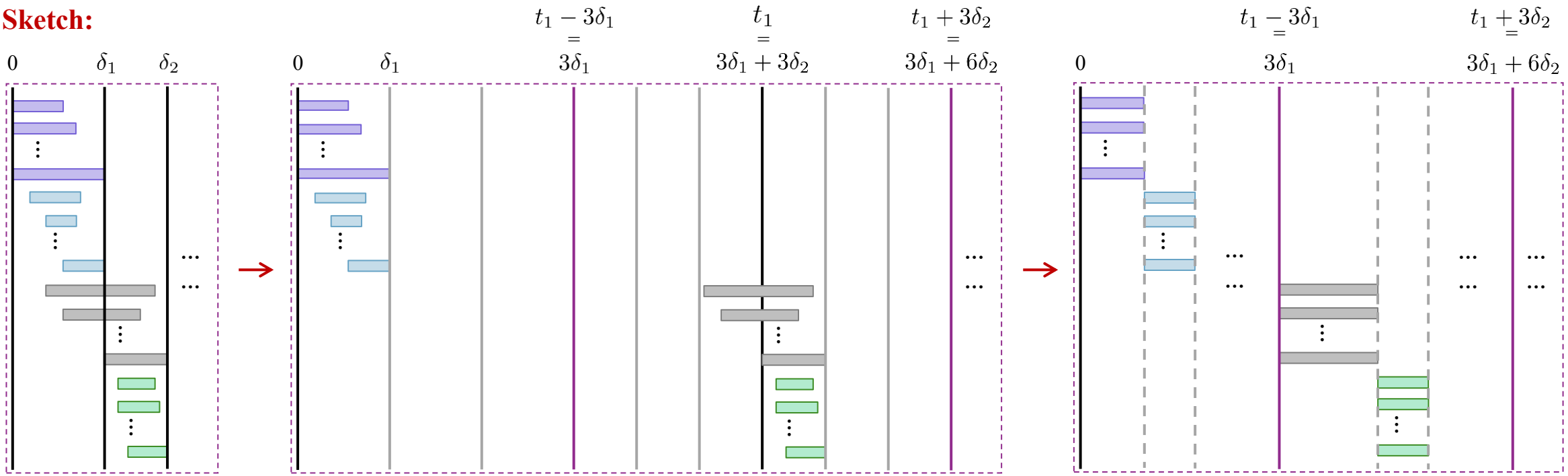
s_i, o_i are small quantities relative to M.

Sketch:



Constant-CR Proof: Step 4, OPT → $\text{OPT}_{\text{transform}}$

Sketch:



In the following paragraphs, we aim to transform the optimal schedule OPT through an iterative process that groups requests into groups $\{\mathcal{Z}_g\}$ with controlled memory utilization. For $g = 1$, let

$$\delta_1 = \max\{p_i + o_i \,|\, r_i \in \mathcal{A}_1\},$$

where $\mathcal{A}_1$ consists of requests initiating at time 0, and

$$\mathcal{Z}_1 = \{r_i \in \mathcal{I} \,|\, p_i + o_i \le \delta_1\}.$$

Subsequently, for $g \ge 2$, let

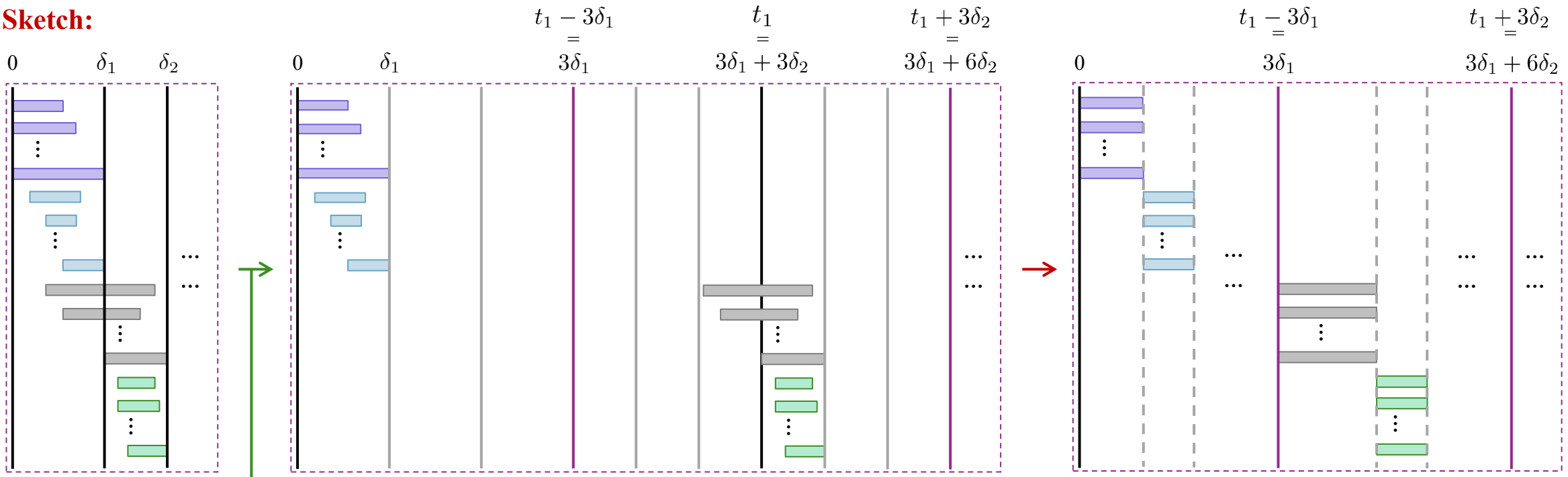
$$\delta_g = \max\{p_i + o_i \,|\, r_i \in \mathcal{A}_g\} - \sum_{j=1}^{g-1} \delta_j,$$

where $\mathcal{A}_g$ consists of requests initiating but not completing by time δ_{g-1}, and

$$\mathcal{Z}_g = \{r_i \in \mathcal{I} \,|\, \sum_{j=1}^{g-1} \delta_j < p_i + o_i \le \sum_{j=1}^{g} \delta_j\}.$$

Constant-CR Proof: Step 4, OPT → $\text{OPT}_{\text{transform}}$

Sketch:



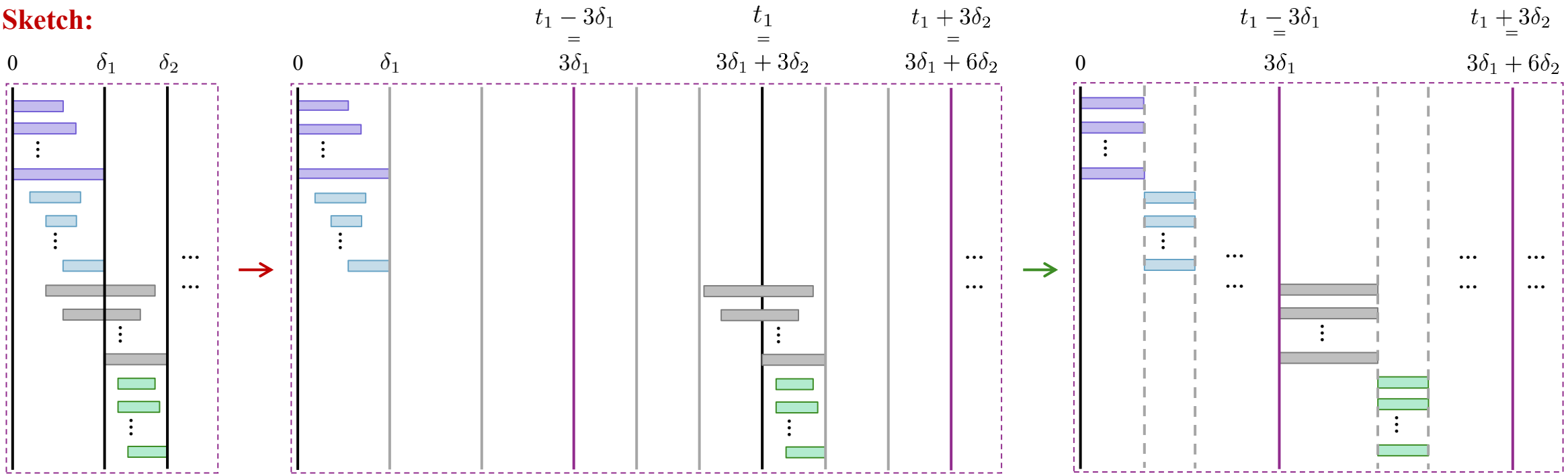
The transformation starts with delay $\mathcal{Z}_g$'s execution by $5\sum_{j=1}^{g-1}\delta_j$ time units for any $g \geq 2$, while preserving the internal timings of all requests in the group. We define the adjusted cutting time for $\mathcal{Z}_g$ as

$$t_g = 6\sum_{i=1}^{g-1}\delta_i,$$

where this transformation scales the original temporal partitioning by a factor of 6.

Constant-CR Proof: Step 4, OPT → $\text{OPT}_{\text{transform}}$

Sketch:



Next, we partition each batch $\mathcal{Z}_g$ into smaller batches $\{\mathcal{W}_h\}$ through the following decomposition:

$$\mathcal{Z}_g = \mathcal{W}_{d_{g-1}+1} + \mathcal{W}_{d_{g-1}+2} + \cdots + \mathcal{W}_{d_g},$$

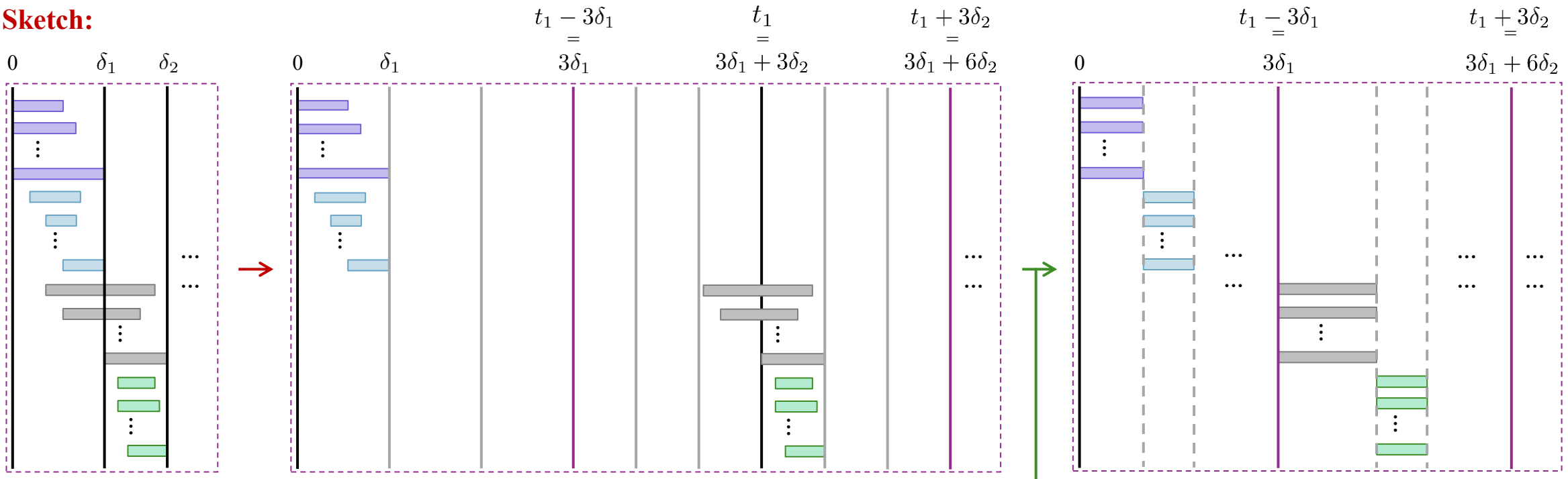
where $d_g = \sum_{j=1}^{g} c_j$ counts the cumulative batches up to group g, with c_j denoting the number of batches in $\mathcal{Z}_j$. The decomposition process constructs $\{\mathcal{W}_{d_{g-1}+k}\}_{k=1}^{c_g}$ as follows: we first sort all requests in $\mathcal{Z}_g$ by their original initiation times.

Then for each subsequent batch, we select the maximal number of remaining requests with the earliest initiation times while respecting the memory constraint M, and iterate this process until all requests in $\mathcal{Z}_g$ are assigned to batches. This way, all the batches except $\mathcal{W}_{d_g}$ must nearly saturate memory M. Now, we replace each request's original response length in $\mathcal{W}_h$ with the batch's average response length:

$$\bar{o}_h = \frac{\sum_{i=1}^{n_h} o_{h,i}}{n_h}.$$

Constant-CR Proof: Step 4, OPT → $\text{OPT}_{\text{transform}}$

Sketch:

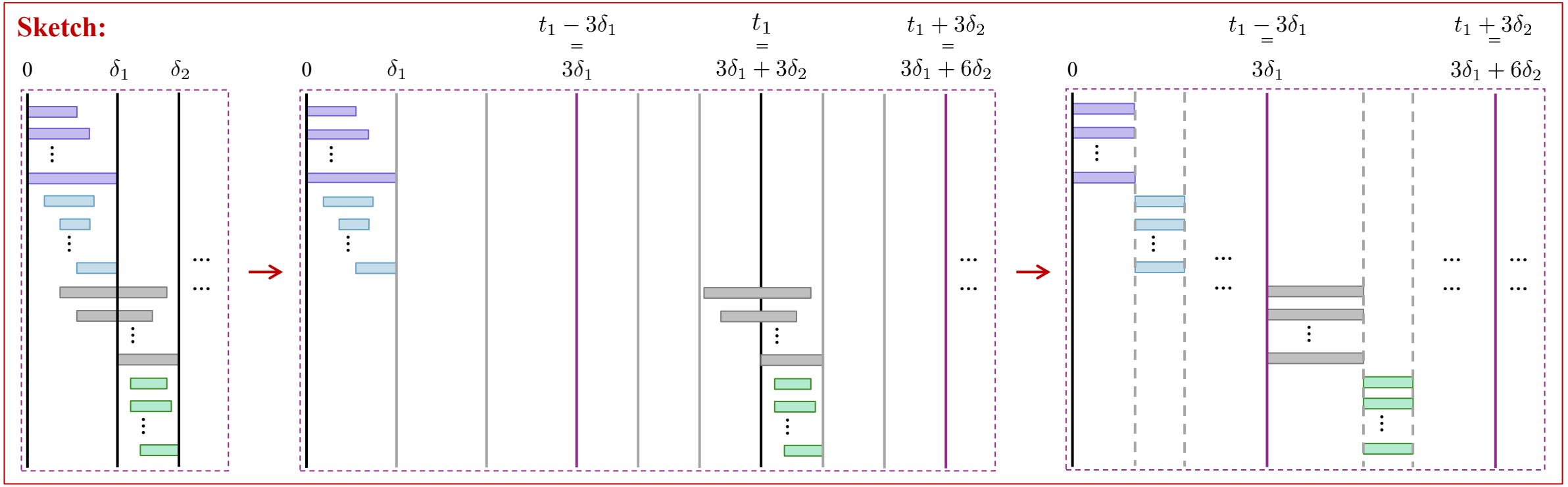


Let $\text{OPT}_{\text{transform}}$ denote the total completion time after this transformation. From inequality (2), we can derive a lower bound on the average memory utilization efficiency η_h for any batch in $\{\mathcal{W}_h\}\backslash\{W_{d_g}\}$:

$$\eta_h \geq \frac{\sum_{r_i \in \mathcal{W}_h} s_i + \sum_{r_i \in \mathcal{W}_h} (s_i + o_i)}{2 \cdot M} > \frac{M - \epsilon(M)}{2 \cdot M} = \frac{1}{2}. \tag{17}$$

Since the original time window for processing $\mathcal{Z}_g$ is less than $\delta_{g-1} + \delta_g$, then by inequality (17), $\mathcal{W}_{d_{g-1}+1}, \mathcal{W}_{d_{g-1}+2}, \cdots, \mathcal{W}_{d_g-1}$ can complete within $2(\delta_{g-1} + \delta_g)$ unit time for any g. Additionally, as the response length of any request in $\mathcal{Z}_g$ is less than $\delta_{g-1} + \delta_g$, $\mathcal{W}_{d_g}$ can complete within $\delta_{g-1} + \delta_g$ unit time. Therefore, $\{\mathcal{W}_{d_{g-1}+k}\}_{k=1}^{c_g}$ can complete within $[t_g - 3\delta_{g-1}, t_g + 3\delta_g]$.

Constant-CR Proof: Step 4, OPT → $\mathrm{OPT}_{\mathrm{transform}}$



Through rigorous derivation (omitted for brievity), we prove that the modified scheduling scheme $\mathbf{ALG}_{\mathbf{group}}$ satisfies the following performance guarantee:

$$\mathbf{TCT}(\mathbf{OPT}_{\mathbf{transform}}) < 6 \cdot \mathbf{TCT}(\mathbf{OPT}).$$

Constant-CR Proof: Step 5, ALG → $OPT_{transform}$

So far, we have already obtained

$$TCT(ALG) \le TCT(ALG_{separate}) \le 4 \cdot TCT(ALG_{group}) \le 8 \cdot TCT(ALG_{align})$$

and

$$TCT(OPT_{transform}) < 6 \cdot TCT(OPT).$$

So the remaining step is to establish the relationship between ALG_{align} and $OPT_{transform}$.

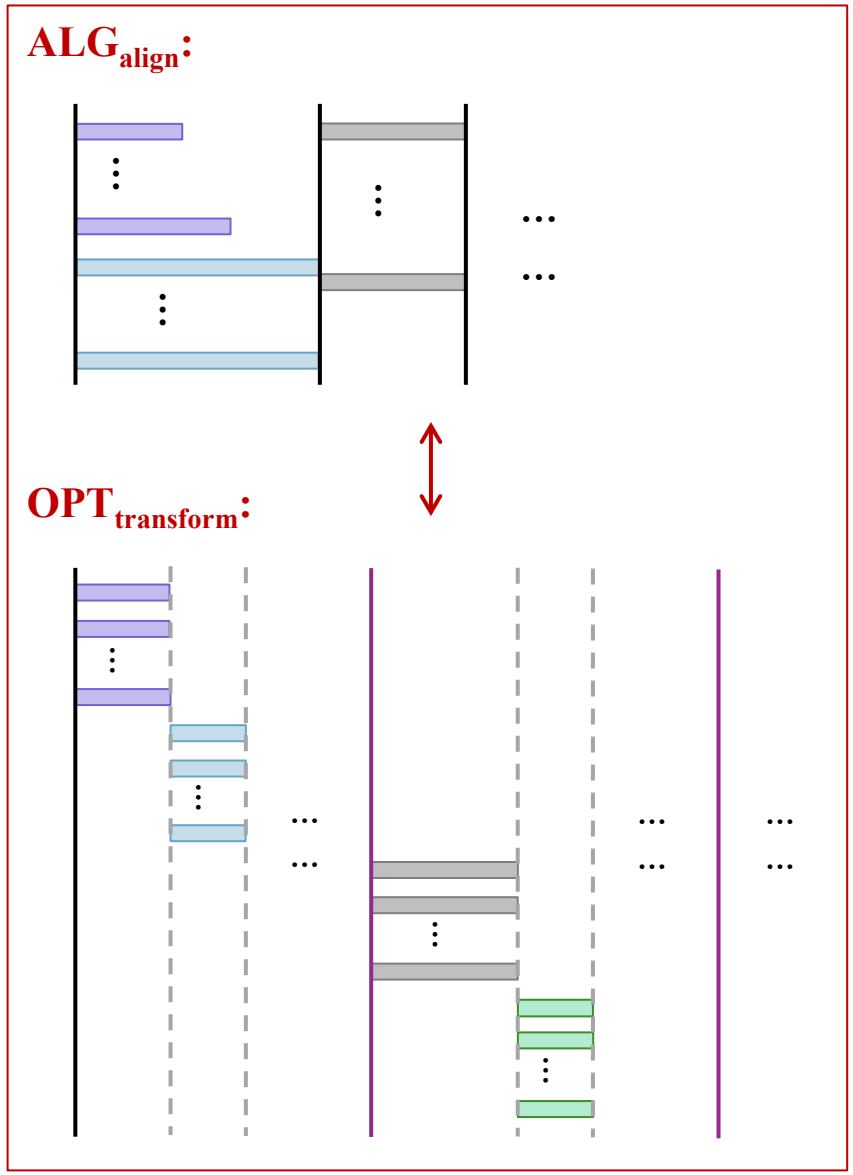
For fixed makespan, minimizing total completion time requires maximizing throughput in the earliest possible time intervals. Formally, our throughput metric

$$\Phi(\mathcal{X}_k) = \frac{n_k}{\bar{o}_k},$$

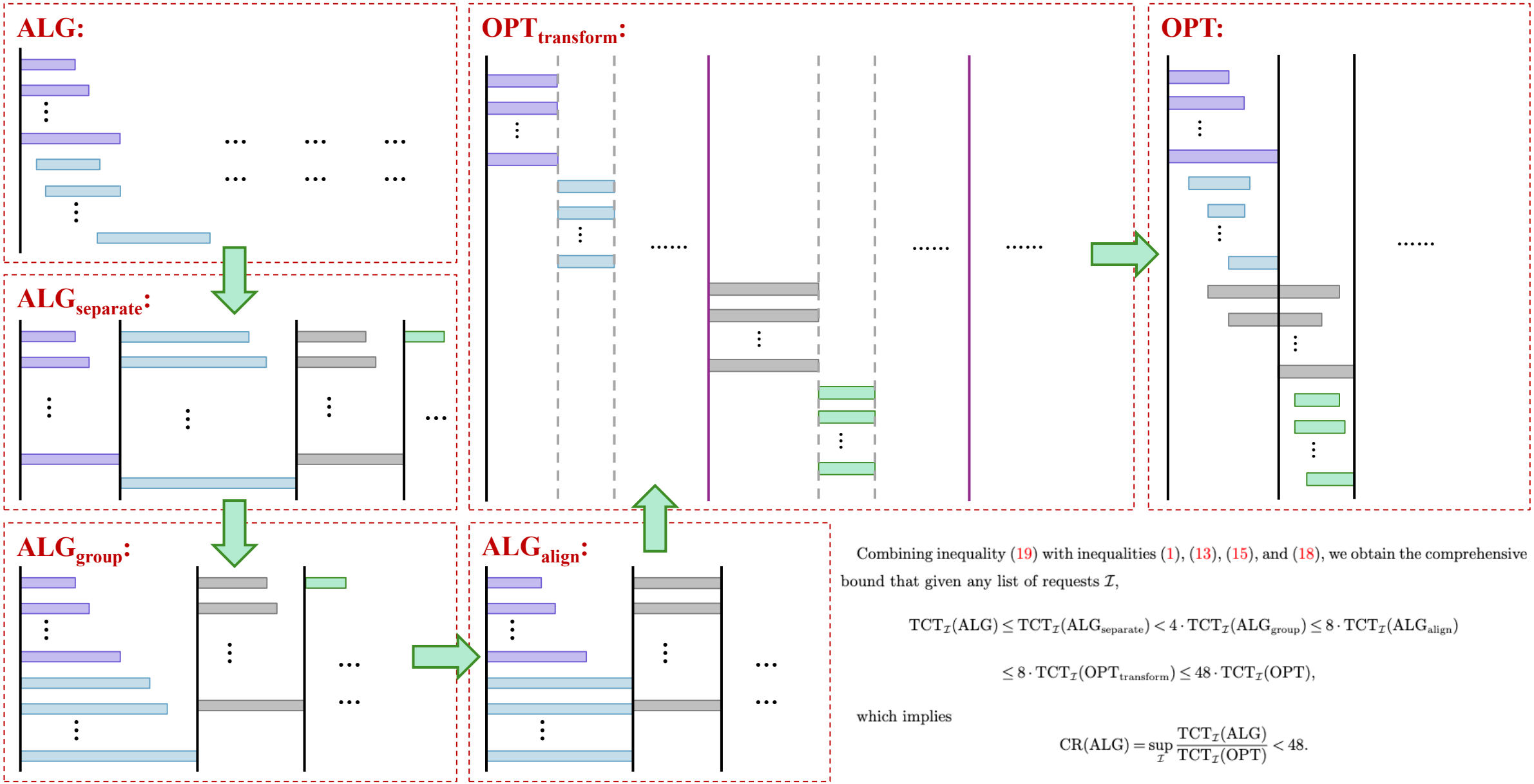
which measures requests completed per unit time in batch $\mathcal{X}_k$, is exactly the reciprocal of our quality metric $F(\mathcal{X}_k)$. This establishes a direct correspondence between our optimization objective and scheduling efficiency.

Inheriting our algorithm's design, ALG_{align} explicitly optimizes for this early throughput maximization by minimizing $F(\mathcal{X}_k)$. For any batch $\mathcal{X}_k$, given fixed prior batches $\mathcal{X}_1, ..., \mathcal{X}_{k-1}$, ALG_{align} minimizes $F(\mathcal{X}_k)$, and thus maximizes $\Phi(\mathcal{X}_k)$. Additionally, each group $\mathcal{Y}_m$ in ALG_{align} achieves an average memory utilization efficiency of $\eta_m > \frac{1}{2}$, while each period $[t_g, t_{g+1}]$ in $OPT_{transform}$ obtains an average memory utilization efficiency of $\eta_{[t_g,t_{g+1}]} \le \frac{1}{6}$. This implies that the makespan of ALG_{align} is strictly less than that of $OPT_{transform}$. This makespan advantage, combined with the early throughput maximization, guarantees that given any list of requests $\mathcal{I}$,

$$TCT_{\mathcal{I}}(ALG_{align}) \le TCT_{\mathcal{I}}(OPT_{transform}). \quad (19)$$



Constant-CR Proof: summary



Combining inequality (19) with inequalities (1), (13), (15), and (18), we obtain the comprehensive bound that given any list of requests $\mathcal{I}$,

$$\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG}) \le \mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG}_{\mathrm{separate}}) < 4 \cdot \mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG}_{\mathrm{group}}) \le 8 \cdot \mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG}_{\mathrm{align}})$$

$$\le 8 \cdot \mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT}_{\mathrm{transform}}) \le 48 \cdot \mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT}),$$

which implies

$$\mathrm{CR}(\mathrm{ALG}) = \sup_{\mathcal{I}} \frac{\mathrm{TCT}_{\mathcal{I}}(\mathrm{ALG})}{\mathrm{TCT}_{\mathcal{I}}(\mathrm{OPT})} < 48.$$

Approximation

In practice, how to select the list of requests that minimizes the quality metric?

Input: $\mathcal{I}$, **Output**: $\mathcal{I}'$

$\mathcal{I}' \leftarrow []$

while $\mathcal{I}$ **do**

  $\mathcal{X} \leftarrow \arg\min_{\mathcal{X} \subseteq \mathcal{I}}(F(\mathcal{X}), -|\mathcal{X}|)$ **subject to** $M(\mathcal{X}, p_i + o_i) \leq M, \forall r_i \in \mathcal{X}$

  Sort $\mathcal{X}$ in ascending order of o_i

  $\mathcal{I}' \leftarrow \mathcal{I}' + \mathcal{X}, \mathcal{I} \leftarrow \mathcal{I} - \mathcal{X}$

end while

Algorithm 1 - Exact Dynamic Programming:

- Optimality: Optimal.
- Complexity: O(n·M), polynomial.
- Shortcoming: Too slow for large M.
- Suggestion on n: $n \leq 100$.

Algorithm 2 - Scaled Dynamic Programming:

- Description: First discretize memory usage, and then perform dynamic programming in scaled space.
- Optimality: $(1 + \varepsilon)$-optimal.
- Complexity: $O(nB/\varepsilon)$, polynomial.
- Suggestion on n: $n \leq 200$.

Approximation

In practice, how to select the list of requests that minimizes the quality metric?

Algorithm 3 – Local Swap Search:

- Description: Initialize a batch with ordered $s_i + o_i$, then iteratively swaps requests to improve the F-metric.

- Optimality: Local-optimal. No global guarantee.
- Complexity: $O(n^2)$, polynomial.
- Suggestion on n: $n \leq 500$.

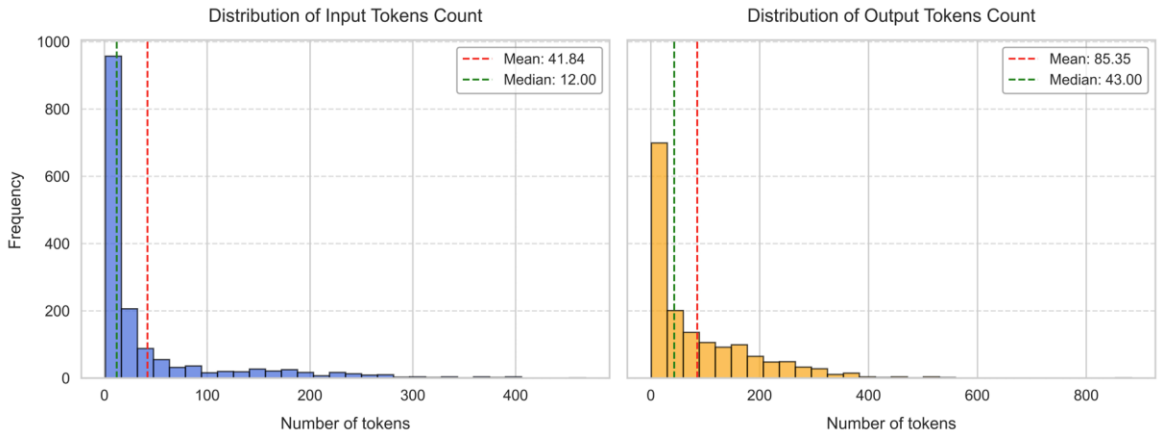
Algorithm 4 – Quantile Greedy:

- Description: Sample a subset of requests and compute quantiles for $s_i + o_i$ and o_i. Select requests below the quantile thresholds. Fill remaining memory with requests sorted by $\frac{o_i}{s_i+o_i}$.
- Optimality: No global guarantee.
- Complexity: $O(n)$, polynomial.
- Suggestion on n: $n \geq 1000$.

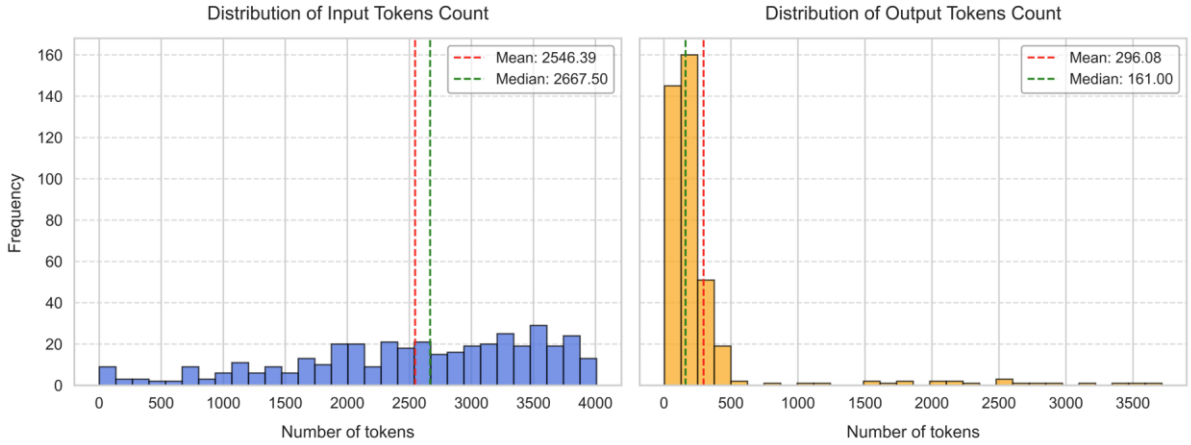
Numerical Experiments

Dataset:

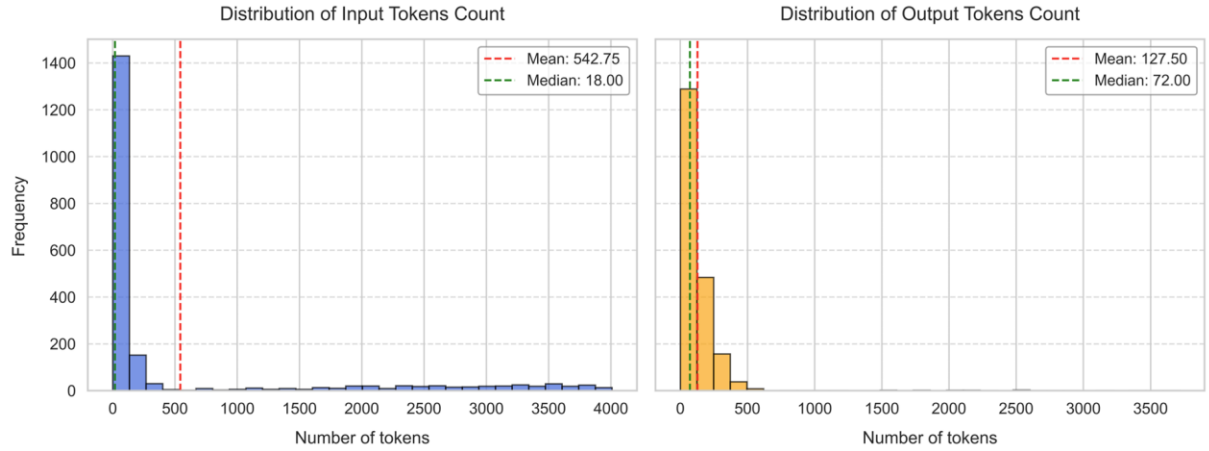
1. Public conversation dataset (Zheng et al. 2023).



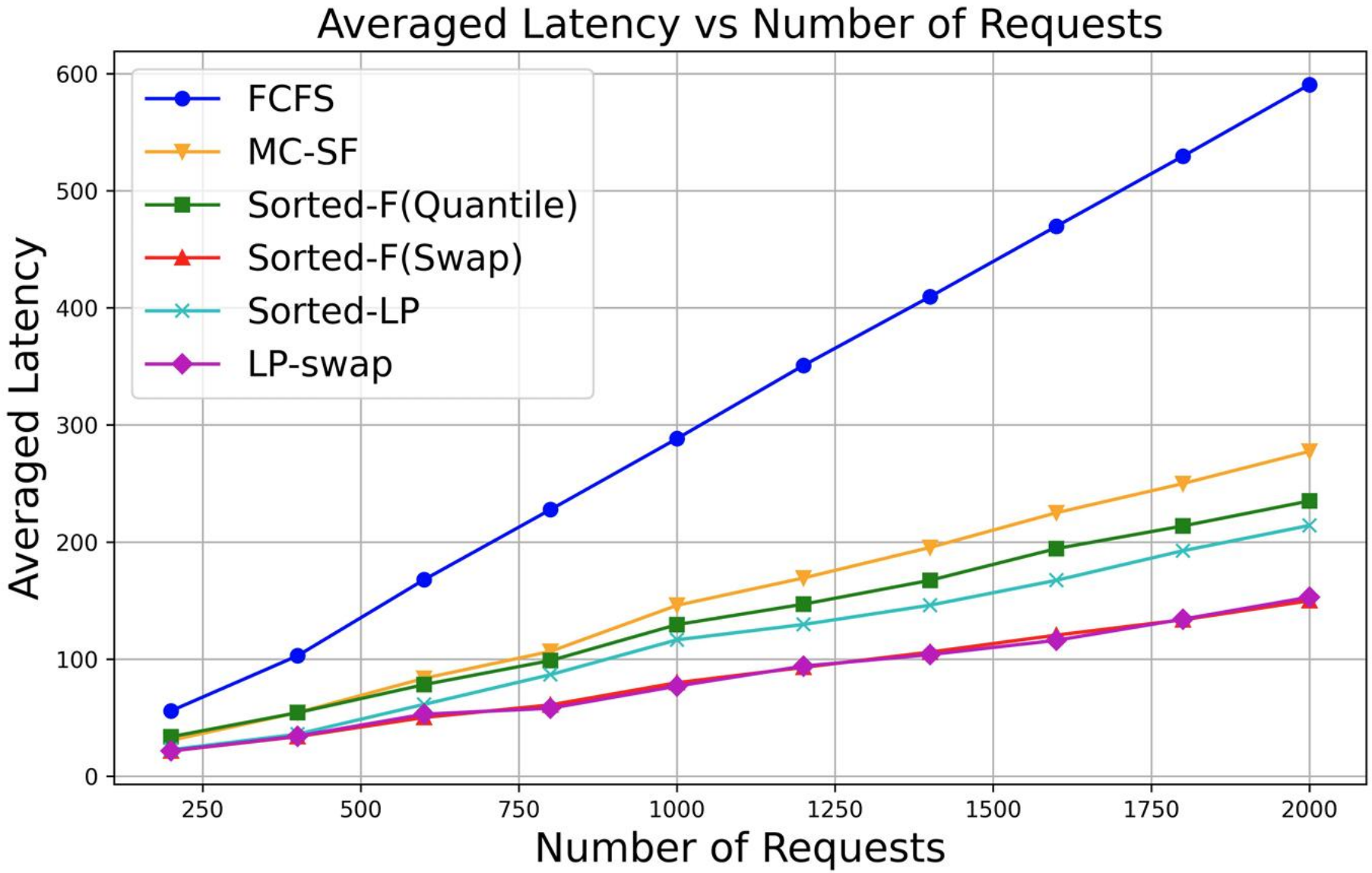
2. Public Arxiv summarization dataset (Cohan et al. 2018).



Mix two datasets:



Numerical Simulation



Robust LLM Inference Scheduling under Output Length Prediction

Zixi Chen (chenzixi22@stu.pku.edu.cn)
Department of Mathematics, Peking University
Yinyu Ye (yyye@stanford.edu)
Department of Management Science and Engineering, Stanford
Department of Industrial Engineering & Decision Analytics, HKUST
Zijie Zhou (jerryzhou@ust.hk)
Department of Industrial Engineering & Decision Analytics, HKUST

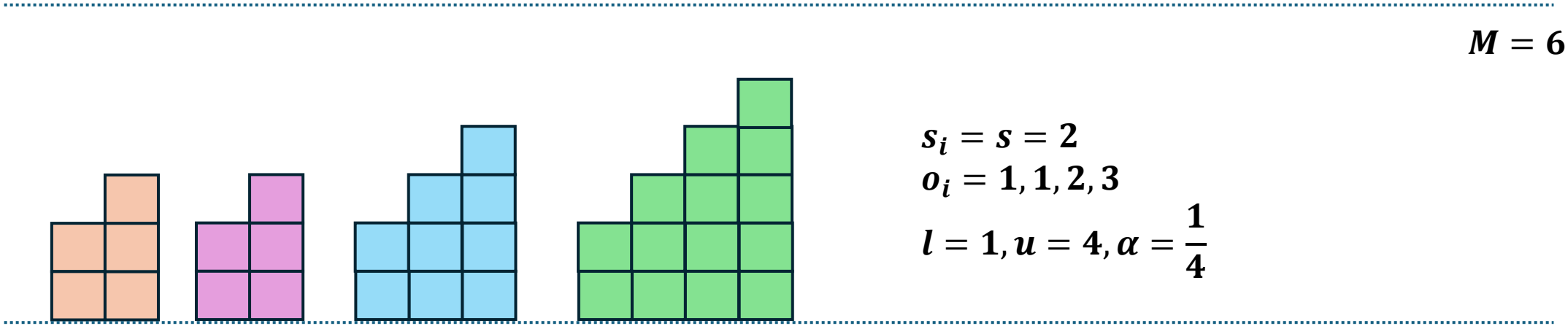
Motivation: Interval Prediction and Classification

- **Quick ML method to classify requests into different groups with disjoint interval prediction.**
  - **E.g. Short (1-200), Medium (201,800), Long (801,1000).**

- **User Input:**
  - **Claude lets user to input the value of min/max tokens in the output when calling the API.**

Model Overview

- Single computational worker with a KV cache of size $\boldsymbol{M}$, capable of storing up to $\boldsymbol{M}$ tokens.
- There are $\boldsymbol{n}$ jobs (prompts), each with size $\boldsymbol{s_i = s > 0}$, known to the decision-maker.
- Let $\boldsymbol{o_i > 0}$ be the realized output length of job i, which is unknown during inference.
- Predictable interval $\boldsymbol{o_i \in [l, u]}$, define $\boldsymbol{\alpha = \frac{l}{u}}$.
- We assume all jobs share the same prediction interval $\boldsymbol{[l, u]}$.



Batch Processing and Memory Constraint

- Jobs processed in batches; each batch takes 1 time unit.
- a_i^t : number of tokens generated for job i at time t. A job is complete when $a_i^t = o_i$.
- S^t : processing requests. Memory used at time t :

$$\sum_{i \in S^t} (s + a_i^t) \leq M$$

- Output lengths are uncertain $\Longrightarrow$ hard to ensure feasibility under non-preemptive policies.
- To handle memory overflow risk, we allow job **cancellation**. Cancelled jobs lose progress and restart from **zero tokens**.

Evaluation Metric: Total End-to-End Latency

- Denote $\boldsymbol{o} = (o_1, \cdots, o_n)$ with $o_1 \leq \cdots \leq o_n$.
- Let I_t be the set of jobs with final start time t.
- Latency for job $i \in I_t$: $L_i = t + o_i$.
- **Total latency**:

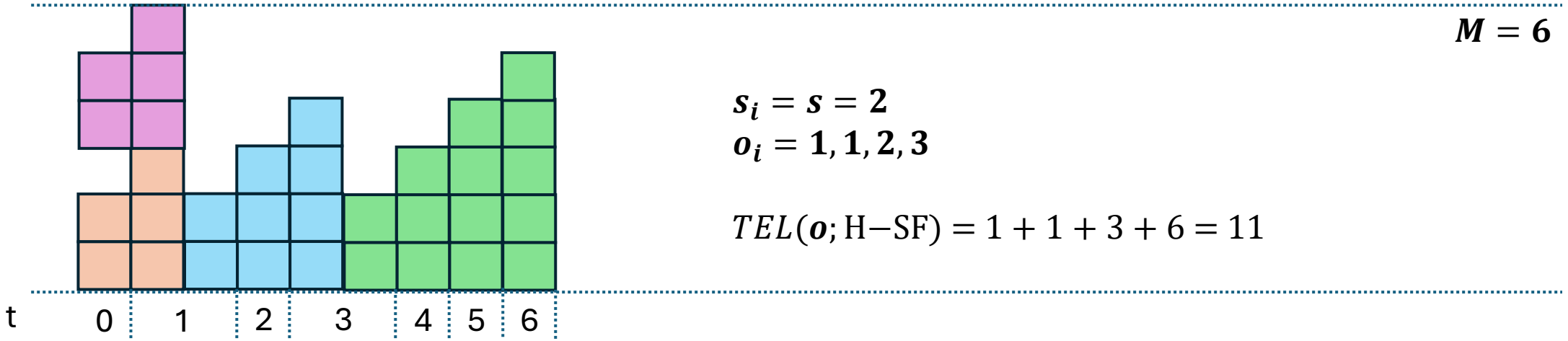
$$TEL(\boldsymbol{o}; \pi) = \sum_i L_i = \sum_{t=0}^{T} t \cdot |I_t| + \sum_{i=1}^{n} o_i$$

Hindsight Benchmark (H-SF)

- **Assumptions:**
  1. All output lengths o_i are **known** in advance.
  2. Follows [Jaillet et al.(2025)] with memory-aware batching and no cancellations.
- **Memory Constraint:** U is the newly added set, S^t denotes the set of jobs already in progress at time t, and p_i is the last start time of job i.

$$\sum_{i \in S^t} (s + t' - p_i)\mathbb{I}_{o_i \geq t' - p_i} + \sum_{i \in U} (s + t' - t)\,\mathbb{I}_{o_i \geq t' - t} \leq M, \qquad \forall t' \in [t, t_{\max}(U)]$$

- **Strategy:**
  1. Choose shortest first (SF).
  2. Pack as many jobs as possible without violating constraint.



$\boldsymbol{M = 6}$

$\boldsymbol{s_i = s = 2}$
$\boldsymbol{o_i = 1, 1, 2, 3}$

$TEL(\boldsymbol{o}; \mathrm{H{-}SF}) = 1 + 1 + 3 + 6 = 11$

Competitive Ratio

- **Definition (Competitive Ratio):**

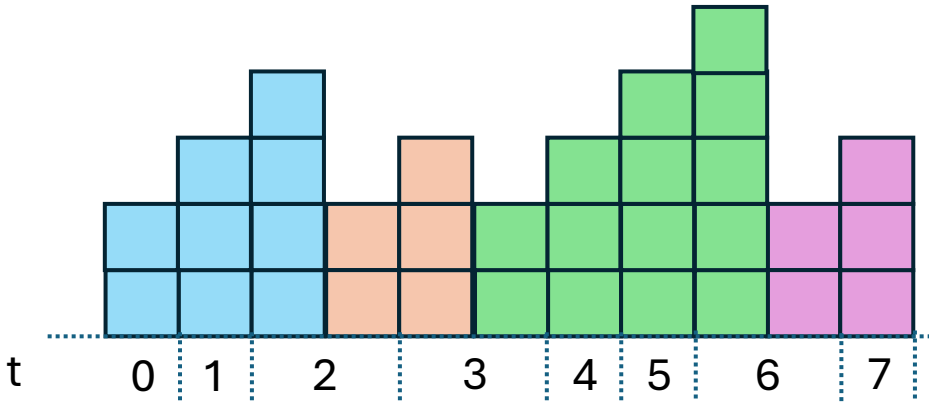
  Let $\boldsymbol{o} = (o_1, \cdots, o_n) \in [\boldsymbol{l}, \boldsymbol{u}]^{\boldsymbol{n}}$ be the true output lengths. For any policy π:

$$CR(\pi) := \sup_{\boldsymbol{o} \in [\boldsymbol{l}, \boldsymbol{u}]^{\boldsymbol{n}}} \frac{\mathbb{E}[TEL(\boldsymbol{o}; \pi)]}{TEL(\boldsymbol{o}; \mathrm{H-SF})}$$

- Measures worst-case ratio to optimal.
- Smaller $CR(\pi)$ means better performance under uncertainty.

Naive Benchmark Algorithm: A_{max}

- **Key Idea**: Assume worst-case output length $\boldsymbol{o_i = u}$ for all jobs.
- **Algorithm Outline**:
  At each time $\boldsymbol{t}$, randomly pick the largest subset $\boldsymbol{U \subset R_t}$ s.t. constraint is met using $\boldsymbol{o_i = u}$.
- **Guarantees feasibility** (no memory overflow).



$\boldsymbol{M = 6}$

$\boldsymbol{s_i = s = 2}$

$\boldsymbol{o_i = 1, 1, 2, 3}$

$\boldsymbol{l = 1, u = 4, \alpha = \frac{1}{4}}$

$TEL(\boldsymbol{o}; \text{H}-\text{SJDF}) = 1 + 1 + 3 + 6 = 11$

$TEL(\boldsymbol{o}; \boldsymbol{A_{max}}) = 2 + 3 + 6 + 7 = 18$

Competitive Ratio of A_{max}

- **Theorem**:
  The competitive ratio of $\boldsymbol{A_{max}}$ satisfies:

$$\frac{\alpha^{-1}\left(1+\alpha^{-1/2}\right)}{2} \leq CR(\boldsymbol{A_{max}}) \leq \frac{\alpha^{-1}(1+\alpha^{-1})}{2} + O(\frac{1}{M})$$

- Overestimation causes under-utilization.
- Lower $\boldsymbol{\alpha = \frac{l}{u}}$ → higher inefficiency.

- The lower bound can be easily checked by giving N_l jobs with $\boldsymbol{o_i = l}$ and N_u jobs with $\boldsymbol{o_i = u}$, where $\frac{N_l}{N_u} = \boldsymbol{\alpha^{-1/2}}$.

- For the upper bound, we introduce a **Memory-Preserving Proof Technique**.

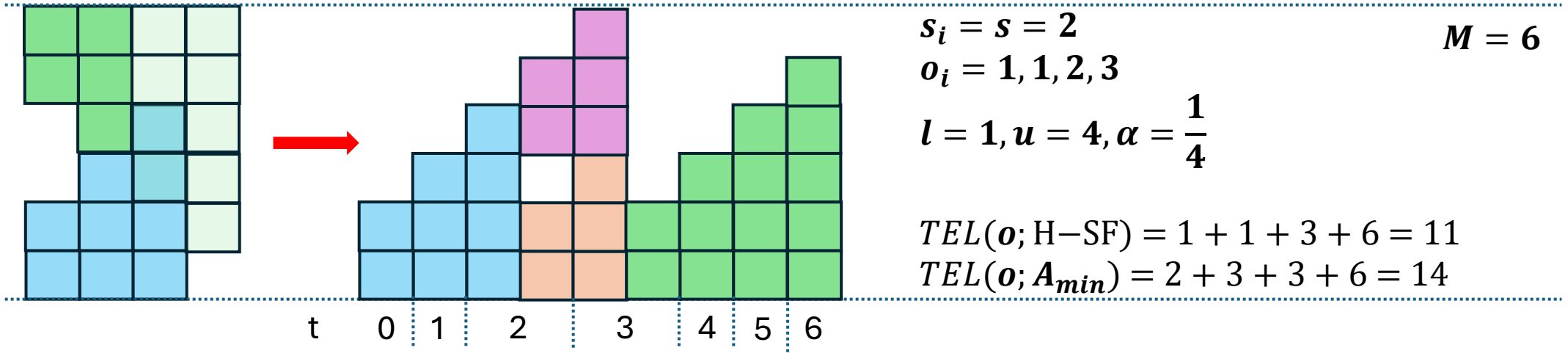
- **Motivation for a Robust Algorithm**
- **Can we design an algorithm that works well under both good and poor predictions?**

Introducing A_{min} : Key Design Principles

- **Key Idea**: Use the lower bound ℓ to estimate memory demand more optimistically. Dynamically refine this bound: set $\tilde{o}_i = \ell$, then increase $\tilde{o}_i$ as output is generated.

- **Algorithm Outline**:
  1. Batch formation: Greedily add requests in order of increasing $\tilde{o}_i$.
  2. Overflow resolution: Delete requests with smallest $\tilde{o}_i$ if memory overflows.

- Robustness arises from being cautious about restarting expensive jobs (with large $\tilde{o}_i$).



$s_i = s = 2$
$o_i = 1, 1, 2, 3$
$l = 1, u = 4, \alpha = \frac{1}{4}$

$M = 6$

$TEL(o; \text{H}-\text{SF}) = 1 + 1 + 3 + 6 = 11$
$TEL(o; A_{min}) = 2 + 3 + 3 + 6 = 14$

Asymptotic Optimality

- **Theorem**: For any online policy π without access to true output lengths,

$$CR(\pi) \geq CR(\boldsymbol{A_{min}})$$

as $M \rightarrow \infty$.

- $\boldsymbol{A_{min}}$ is asymptotically optimal among policies with limited output knowledge. No policy can uniformly outperform it.

Theoretical Guarantee: Robust Competitive Ratio

- **Theorem**: The competitive ratio of $\boldsymbol{A_{min}}$ equals to the following Rayleigh Quotient:

$$CR(\boldsymbol{A_{min}}) = \max_{\boldsymbol{x}>0} \frac{\boldsymbol{x}^T A_u \boldsymbol{x}}{\boldsymbol{x}^T B_u \boldsymbol{x}} + O(\frac{1}{M}).$$

- $A_u = (a_{ij})_{u*u}, a_{ij} = \frac{\min(i,j)^2}{ij}(\frac{1}{2}(i+j)^2 - \min(i,j)^2).$
- $B_u = (b_{ij})_{u*u}, b_{ij} = \min(i,j)^2.$

- **Proposition**: The Rayleigh quotient is uniformly bounded by $O(\log u) = O(\log(\alpha^{-1}))$.

- When $\alpha = 1$, $CR(\boldsymbol{A_{min}}) = 1$, optimal under perfect prediction.

- When $\alpha \to 0$, $CR(\boldsymbol{A_{min}})$ increases in a log scale.

Scheduling under Disjoint Interval Predictions

- We further relax the assumption that the prediction intervals $[\boldsymbol{l}, \boldsymbol{u}]$ of each $\boldsymbol{o_i}$ are the same.
- we now generalize the model to allow $\boldsymbol{m}$ disjoint prediction intervals $[\boldsymbol{l_1}, \boldsymbol{u_1}], [\boldsymbol{l_2}, \boldsymbol{u_2}], \ldots, [\boldsymbol{l_m}, \boldsymbol{u_m}]$, where $\boldsymbol{u_i} < \boldsymbol{l_{i+1}}$ for all $\boldsymbol{i} \in [\boldsymbol{m-1}]$.
- $\boldsymbol{A_{min}}$ is modified to initialize $\tilde{\boldsymbol{o}}_{\boldsymbol{i}} = \boldsymbol{l_j}$ if it belongs to the $\boldsymbol{j}$th interval. It continues to prioritize shorter requests while maintaining its robustness and adaptive batching strategy.
- **Theorem**: The competitive ratio of $\boldsymbol{A_{min}}$ is bounded by:

$$CR(\boldsymbol{A_{min}}) \leq O(\log(\alpha^{-1})) + O(\frac{1}{M}).$$

Moreover, for any online policy π without access to true output lengths,

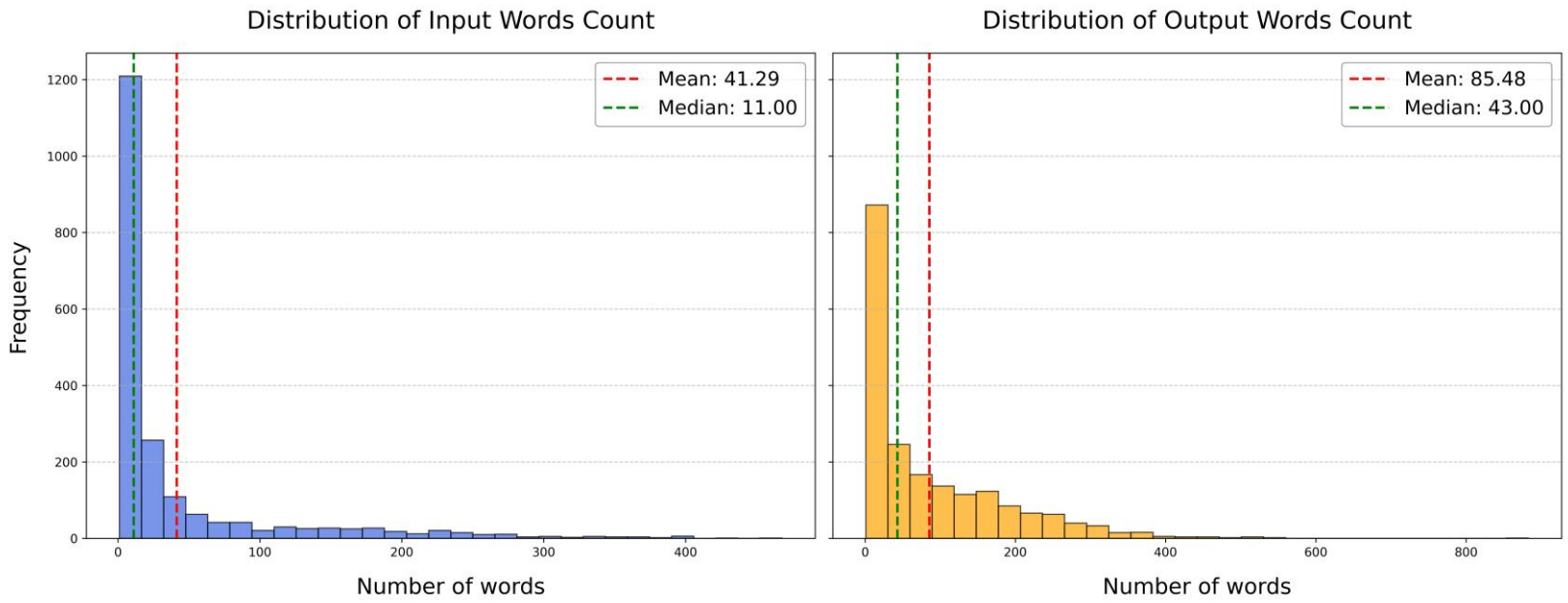
$$CR(\pi) \geq CR(\boldsymbol{A_{min}})$$

as $M \to \infty$.

Numerical Experiments

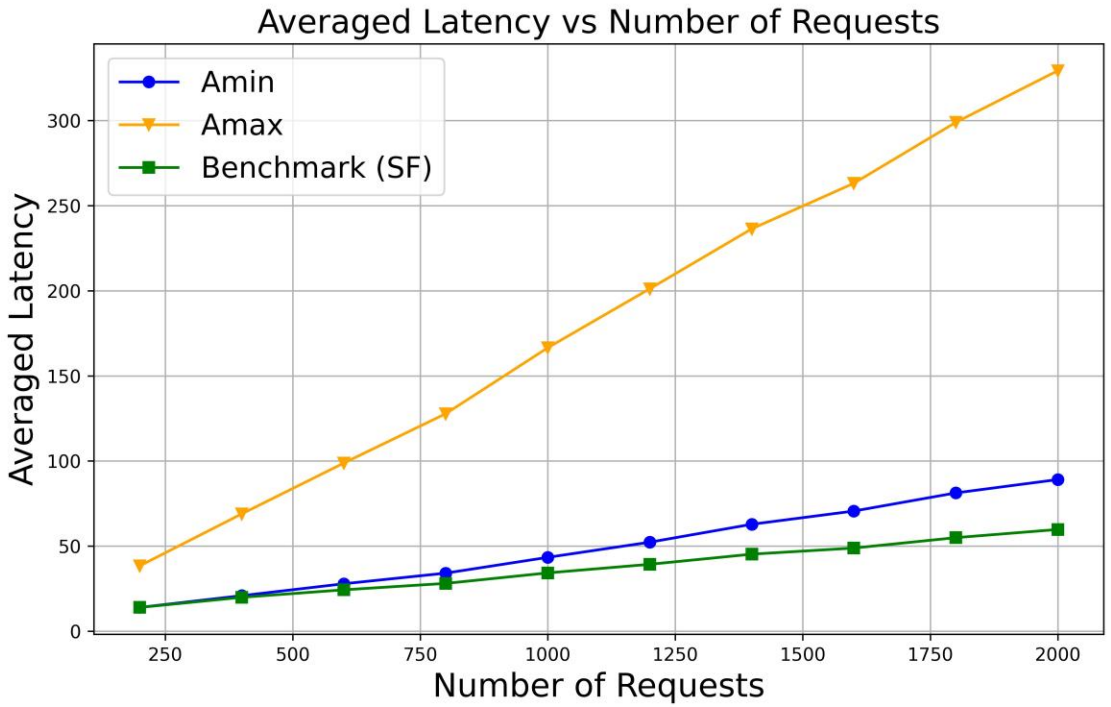
- **Dataset Overview.**
- 2,000 conversations (Zheng et al. 2023)

- The input sizes range from 1 to 468 tokens, with a mean of 41, median of 11, and variance of 4,961.
- The output lengths range from 1 to 883 tokens, with a mean of 85, median of 43, and variance of 9,702



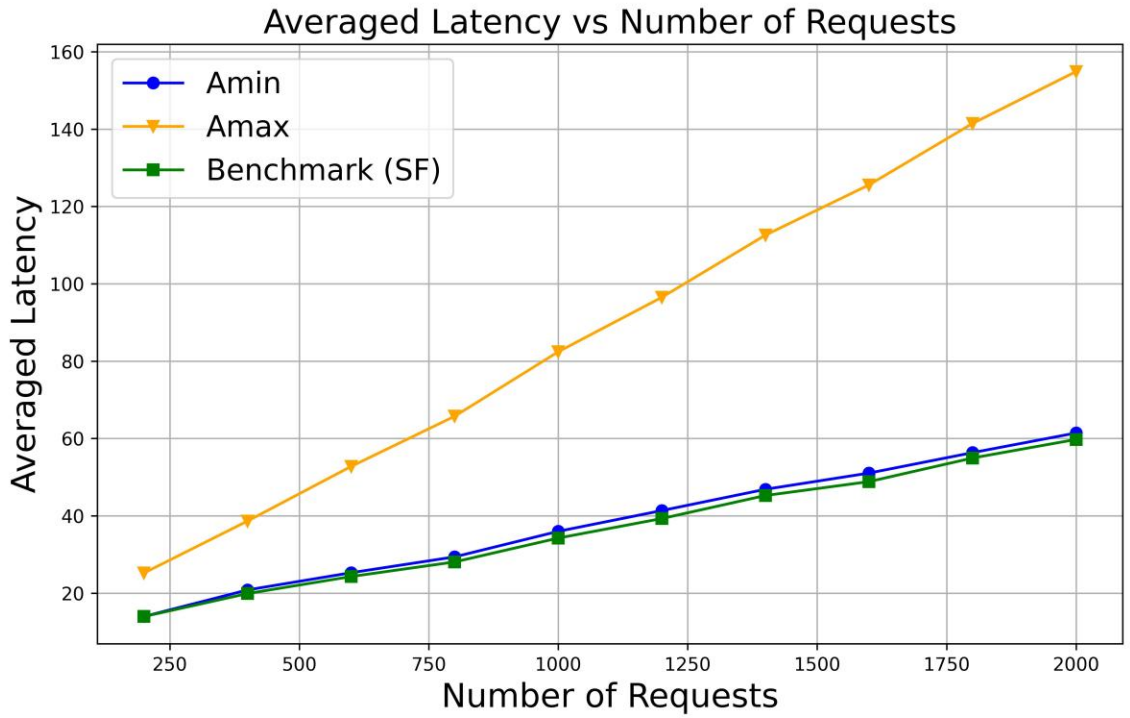
Rough Prediction

- Each request has a coarse prediction interval of **[1, 1000].**
- Representing **minimal information** about the output length.
- A_{max} : poor memory utilization.
- A_{min} : achieves average latency nearly identical to the benchmark.



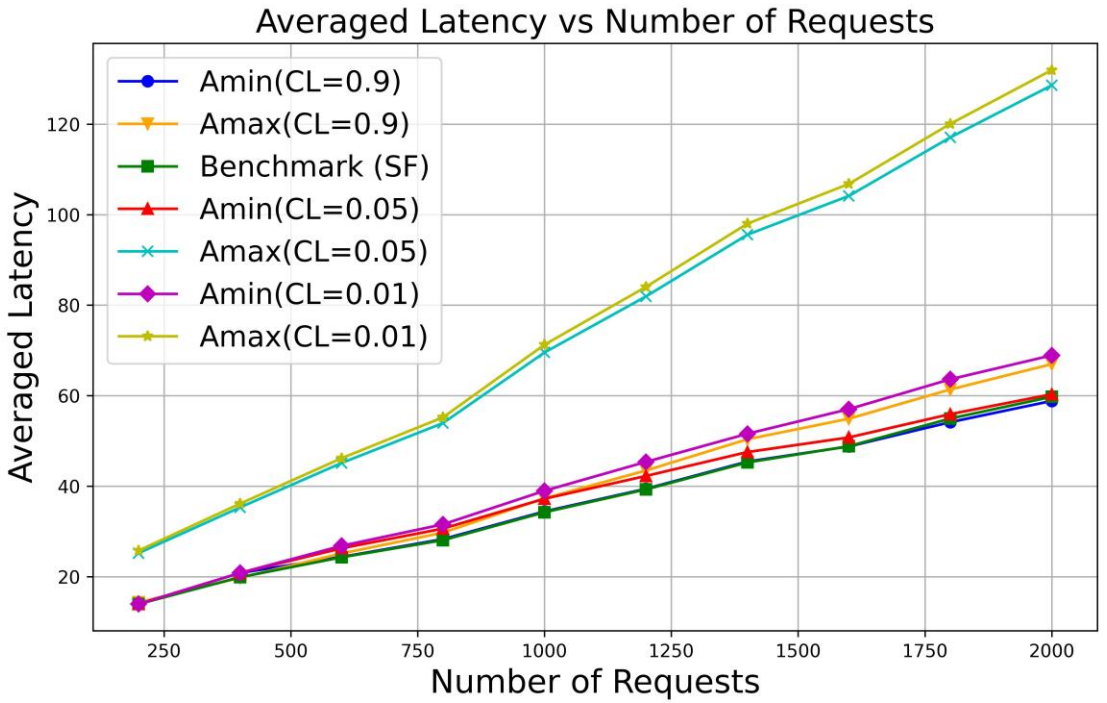
Non-Overlapping Classification

- We assign each request's prediction interval to one of the fixed buckets: **[1, 100], [101, 200], . . . , [901, 1000].**
- Simulating a **multi-class** classification model with **non-overlapping** intervals.
- A_{max} : achieves a substantial improvement over its performance in Experiment 1.
- A_{min} : closely approaches that of the benchmark.



Overlapping Interval Prediction

- Each request $\boldsymbol{i}$ is assigned a prediction interval of the form $[(\mathbf{1}-\boldsymbol{x})\boldsymbol{o_i}, (\mathbf{1}+\boldsymbol{x})\boldsymbol{o_i}]$, where $\boldsymbol{x} \in \{\mathbf{0.1}, \mathbf{0.95}, \mathbf{0.99}\}$.
- Larger values of x correspond to more precise predictions.
- $\boldsymbol{x} = \mathbf{0.1}$, indicating highly accurate predictions, both Amax and Amin perform well
- $\boldsymbol{x} = \mathbf{0.95}$ or $\boldsymbol{x} = \mathbf{0.99}$, the performance of $\boldsymbol{A_{max}}$ deteriorates significantly. $\boldsymbol{A_{min}}$ continues to maintain low average latency even under highly uncertain predictions.



Thank you!

Contact email: jerryzhou@ust.hk