

A Analytical estimate of PipelineRL speedup for fixed max lag

In this additional section we estimate how much faster PipelineRL can be compared to Conventional RL for the same value of maximum token lag g_{max} . We will be using the following notation, mostly the same as in the main text:

- N is the number of accelerators
- $S = BG$ is the number of sequences that are processed in each Conventional RL step
- L is the maximum and \bar{L} is the average sequence length for the current policy π
- $K = S\bar{L}$ is total number of tokens that Conventional RL processes in each optimizer step

We will additionally use $U(h)$ to refer to the accelerator’s maximum flops utilization when running typical Transformer kernels at batch size h .

A.1 Units

To compare throughputs of different RL approaches it useful to adopt time and throughput units that don’t depend on the particular GPU model and the LLM size. To this end we introduce a time unit called *flash*:

$$f = \frac{F_{gen}}{M} \quad (8)$$

where F_{gen} is the number of FLOPs required for one token forward pass for the chosen LLM, and M is the maximum theoretical FLOPs throughput for the given GPU. The meaning of a flash is the theoretically smallest amortized time that a token generation can take. Thus generating K tokens will take at least K flashes, though at a more typical generation utilization of ~ 0.1 rate it will take $10K$ flashes. For very long sequences F_{gen} can vary significantly due to attention FLOPs becoming a large part of total FLOPs, but for simplicity here we will abstract away from this detail.

Having introduced flash f as the unit, we will measure the system throughput in *tokens per flash*.

Let τ be the amortized training time per token. τ will be similar at scale for PipelineRL and Conventional RL, because both approaches can benefit from sequence packing.

A.2 Conventional RL throughput

We can express Conventional RL throughput as follows:

$$r_{conv} = \frac{K}{t_{conv}^{gen} + t_{conv}^{train}}, \quad (9)$$

where t_{conv}^{gen} and t_{conv}^{train} are times that generation and training take respectively. Let’s look at these terms closer:

$$t_{conv}^{gen} = \sum_{l=1}^L \frac{h(l)/Nf}{U(h(l)/N)} \quad (10)$$

$$t_{conv}^{train} = \frac{K\tau}{N} \quad (11)$$

where $h(l)$ is the number of sequences still in progress after l steps of decoding, and $U(h)$ is the GPU utilization at batch size h . To understand Equation (10), recall that generating k tokens by definitions takes k flashes under perfect GPU utilization and $k/U(k)$ at the utilization $U(k)$.

We can rewrite this in terms of tokens / flash throughputs:

$$r_{conv} = \frac{1}{\frac{1}{r_{conv}^{gen}} + \frac{1}{r_{conv}^{train}}} \quad (12)$$

$$r_{conv}^{gen} = \frac{K}{\sum_{l=1}^L \frac{h(l)/N}{U(h(l)/N)}} \quad (13)$$

$$r_{conv}^{train} = \frac{N}{\tau} \quad (14)$$

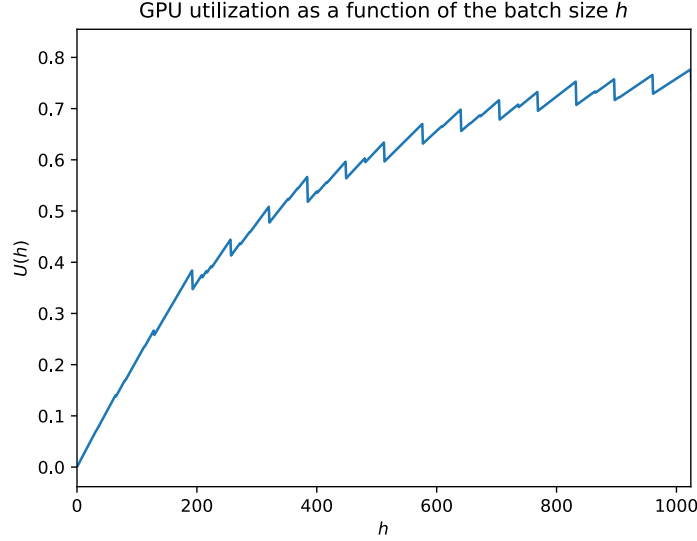


Figure 7: H100 utilization at batch size h as the ratio of maximum theoretical bf16 FLOPS throughput. We use $(4096, h) \cdot (h, 16384)$ matrix multiplications for the measurement. For every h we consider padding up to $h + 64$ to increase the speed, because empirically we observed large utilization bumps when h is divisible by a higher power of 2 (up to 128).

At low batch size per GPU at step l , $h_N(l) = h(l)/N$, the ratio $h_N(l)/U(h_N(l))$ will only decrease very slowly as a function of N , because for modern GPUs $\frac{x}{U(x)}$ is nearly constant for small x . This is the formal explanation for Conventional RL’s decreasing efficiency as N grows.

The maximum token lag in the setup we described above is $S - 1$.

A.3 PipelineRL throughput

For PipelineRL the system throughput is determined by the slowest pipeline stage. Using the concepts introduced above, the throughput of PipelineRL can be estimated as follows:

$$r_{\text{pipeline}} = \min(r_{\text{pipeline}}^{\text{gen}}, r_{\text{pipeline}}^{\text{train}}) \quad (15)$$

$$r_{\text{pipeline}}^{\text{gen}} = U(H)I \quad (16)$$

$$r_{\text{pipeline}}^{\text{train}} = \frac{N - I}{\tau} \quad (17)$$

To understand the maximum lag of Pipeline RL consider the fact that the generation GPUs will produce HIL tokens during the time it takes to generate the longest possible sequence of length L . On average there will be $\frac{HIL}{L}$ sequences in these tokens. Thus, in the worst case when an optimizer step happened just before the longest sequence generation started, a long sequence will be used for training $g_{\max} = \lceil \frac{HIL}{LB} \rceil$ optimizer steps later than its generation started.

To build a same-lag equivalent for a conventional RL system, one needs to maximize $r_{\text{pipeline}}(H, I)$ while keeping $\lceil \frac{HIL}{LB} \rceil \leq S - 1$. We could found this problem difficult to solve analytically, and performed a straight-forward search of all (H, I) configurations for our investigations below.

A.4 A PipelineRL speedup case study

To compute the exact throughput boost that PipelineRL brings it is necessary to make assumptions about the sequence length distribution and the hardware that is used for the experiments. For the case-study below, we assume uniform length distribution from 1 to the max length L and H100 as the GPU. We visualize the GPU utilization table $U(h)$ in Figure 7. The reader can see that $U(h)$ grows almost linearly up to $h \sim 200$, which makes it possible to compress the generation on fewer GPUs

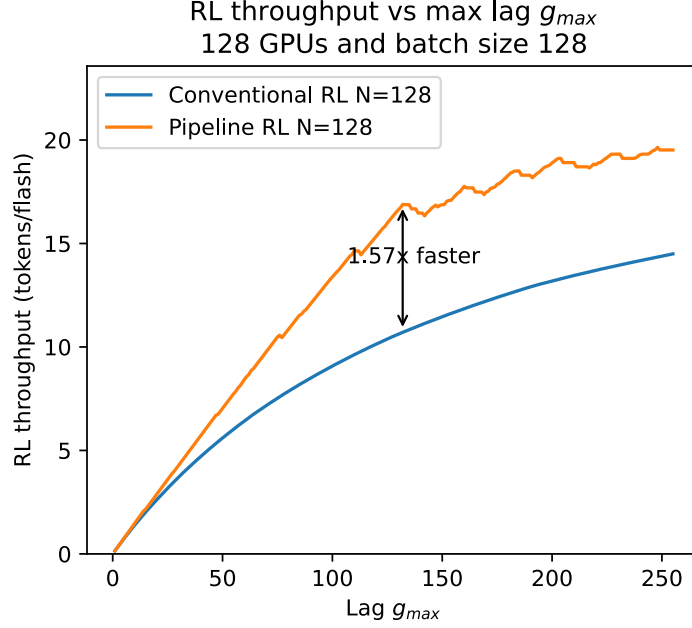


Figure 8: Pipeline RL and Conventional RL throughputs as the function of the maximum lag g_{max} for a setup with $N = 128$ GPUs and batch size $B = N$.

at a higher utilization. For a setup with $N = 128$ GPUs and training batch $B = 128$ we considered all possible (I, H) configurations of PipelineRL and plotted their throughput as a function of the lag g_{max} . Figure 8 shows that PipelineRL can be up to $1.57x$ faster for $g_{max} \sim 133$. This lag value can be too high for many practical setups, but with a higher batch size of e.g. $B = 2048$ the same number of sequences to be generated by each GPU will correspond to a practical $16x$ lower lag $g_{max} \sim 8$.

The mechanics of how PipelineRL achieved the improvement are as follows:

- $r_{pipeline}^{gen} = 16.9, r_{pipeline}^{train} = 17.08, \mathbf{r}_{pipeline} = \mathbf{16.9}, H = 192, I = 44$
- $r_{conv}^{gen} = 18.3, r_{conv}^{train} = 26.02, \mathbf{r}_{conv} = \mathbf{10.7}$

Clearly, the root cause of PipelineRL's speedup is that the 44 generation GPUs can produce 16.9 tokens per flash, that is more efficient than having 128 GPUs produce 18.3 tokens per flash in the Conventional RL case.