Appendix

A Additional Instruction Tuning Implementation Details

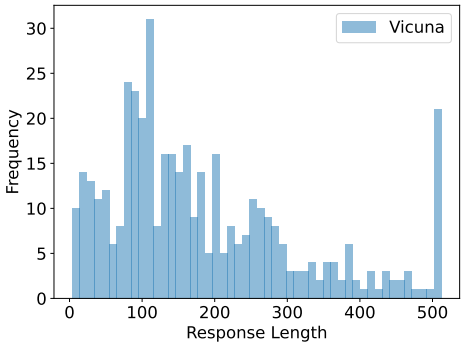
To fine-tune the instruction-tuned length predictor module, we followed a specific procedure. First, we prepared the dataset by sampling each instruction four times. One sample was obtained using greedy decoding with a temperature of 0, while the other three samples were generated using a temperature of 1.0 with different random seeds. The maximum length among the four samples was used as the target length for each instruction.

For the training dataset, we constructed new instructions by appending the requirement of perceiving the response length only. The prompt we used for this purpose was: "Don't output the response for the above instruction. Instead, you need to predict the number of tokens in your response. Output only one number."

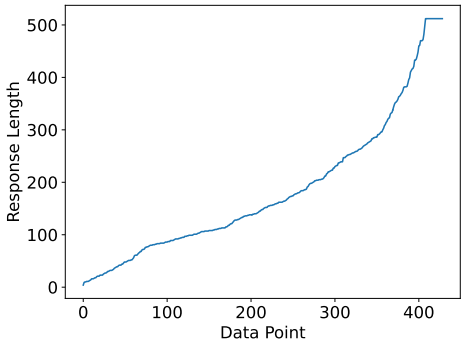
During the training process, we employed the same hyperparameters as the Vicuna [4] instruction tuning process on LLaMA [34]. Specifically, we set the learning rate to 0.00005 and trained the model for three epochs. We applied the LoRA [16] method solely to the query and key linear layer. The training was conducted on a single 80GB A100 GPU. All codes are implemented in PyTorch [26].

B Distribution of Instruction-in-Wild Dataset

The histogram of response length on Instruction-in-Wild [36] dataset is shown in Figure 5. The response length is more diverse and contains more responses with a long length.



(a) Response length distribution of Instruction-in-Wild dataset.



(b) Distribution of response length on instruction-in-Wild dataset per sample.

Figure 5: Distribution of response length on instruction-in-Wild dataset.

C Discussion on Sequence Scheduling with PiA

In the main text, we presented the sequence scheduling technique using instruction-tuned models, where the LoRA weight was utilized for response length perception. However, recent LLMs such as GPT-4 and Claude have shown promising performance in Perception in Advance (PiA), which allows them to leverage PiA for sequence scheduling without the need for additional weights.

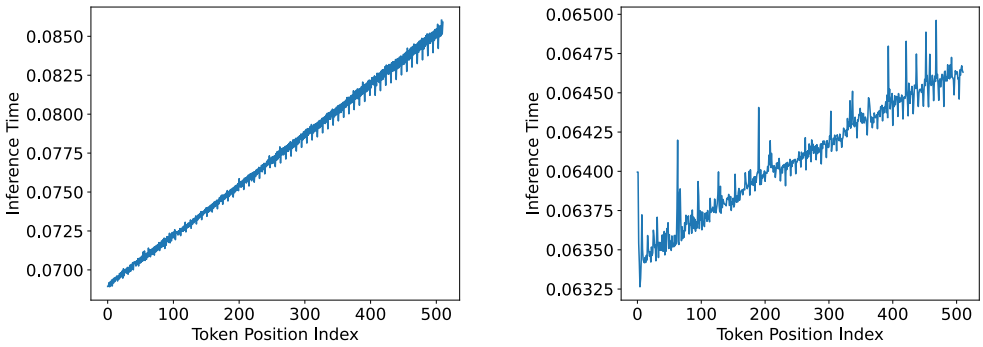
To further improve the speed of inference in this setting, one potential approach is to reuse the key-value (kv) cache of input queries from the response length perception stage during the inference stage. This eliminates the need for recomputing the kv-cache on input instructions, thereby saving valuable processing time.

One strategy we explored is offloading the kv-cache to the main memory and then loading it back into the GPU memory. However, the transfer time between the CPU and GPU can be significant and greatly impact overall performance, often surpassing the time saved by recomputation. To address this, one possible improvement is to offload and load only a portion of the kv-cache asynchronously, reducing the transfer time overhead. This is an area that we leave for future work and exploration.

Another approach we investigated involved compressing the kv-cache and storing it directly in the GPU memory. We applied FlexGen's quantization method [29] for compression and found that it had minimal impact on performance. However, this approach does consume additional memory and can lead to smaller batch size, potentially degrading overall performance. A potential avenue for further exploration is to combine compression and offloading techniques to strike a balance between memory usage and performance.

Considering these factors, we have chosen to continue using the recomputation strategy for the PiA response length perception module in our proposed pipeline. While there is potential for optimizations through offloading and compression, further investigation and refinement are required to achieve substantial performance gains in practice.

D Inference Time Grows with Token Position Index



(a) batch-size=16

(b) batch-size=1

Figure 6: Inference time grows with the token position index.

In transformer models, the inference time for tokens located toward the end of a sequence is typically longer due to the need for self-attention operations on more keys and values. This results in a linear increase in the inference time required for tokens as the location index grows, as illustrated in Figure 6. Consequently, saving redundant computations on longer sequences has a more significant impact compared to shorter ones. This observation explains why the growth ratio of throughput can be higher than the ratio of saved redundant tokens.