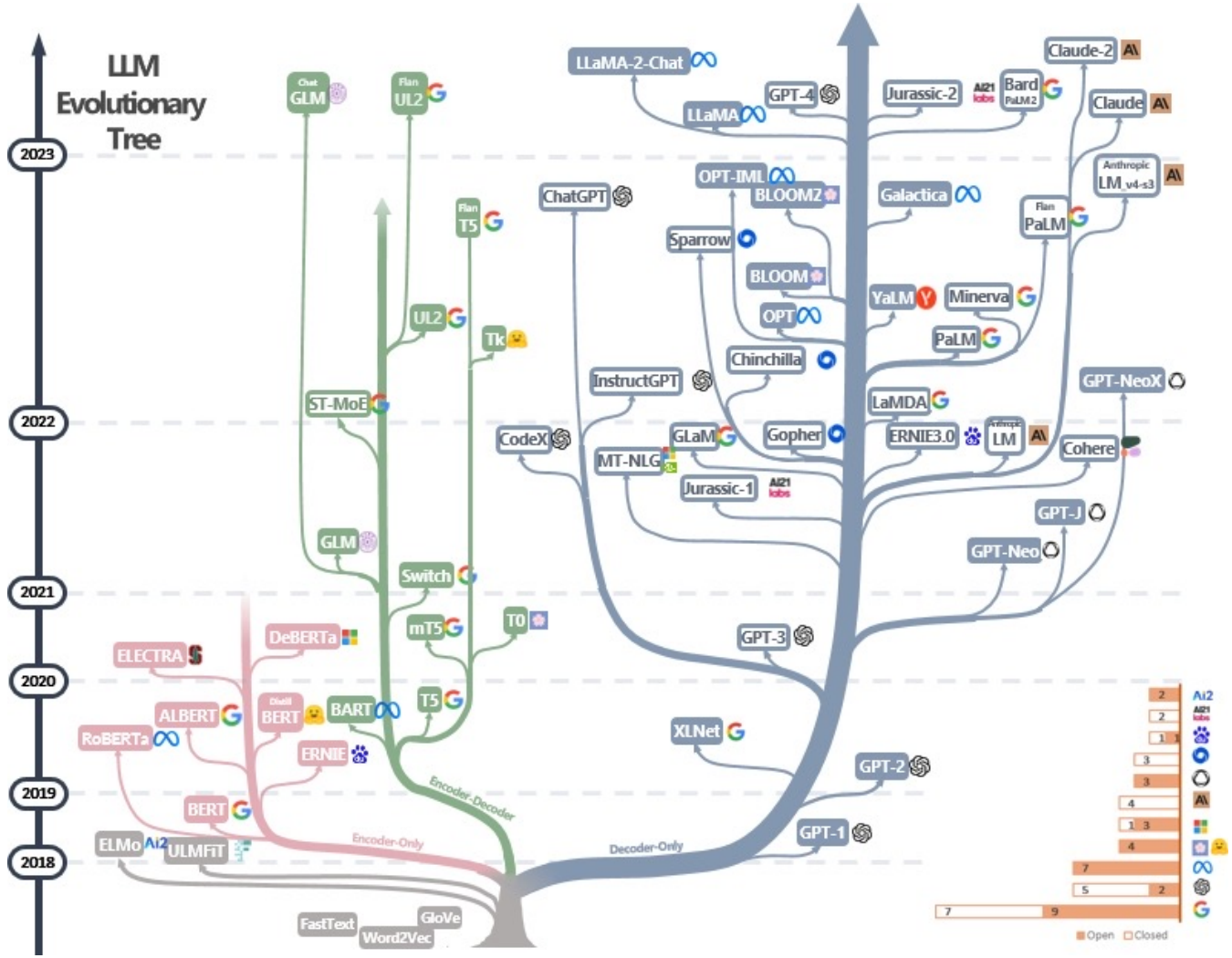
hLLM: A Numa-aware Heterogeneous Platform for High-throughput Large Language Models Service

Kexin Chu Tzechinh Liu Pengchao Yuan Wei Zhang

University of Connecticut

LLM Evolutionary Tree



[Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond]

Cost of Serving

OpenAI GPT API Pricing Calculator

Wondering about how the OpenAI GPT and other AI model pricing works? Here's a pricing calculator for OpenAI GPT API, Azure OpenAI APIs, Anthropic Claude API.

OpenAI gpt-3.5-turbo | OpenAI gpt-3.5-turbo-instruct | OpenAI gpt-4-turbo (beta) | OpenAI gpt-4 | OpenAI gpt-4-32k | Anthropic claude-2 | Anthropic claude-instant-1

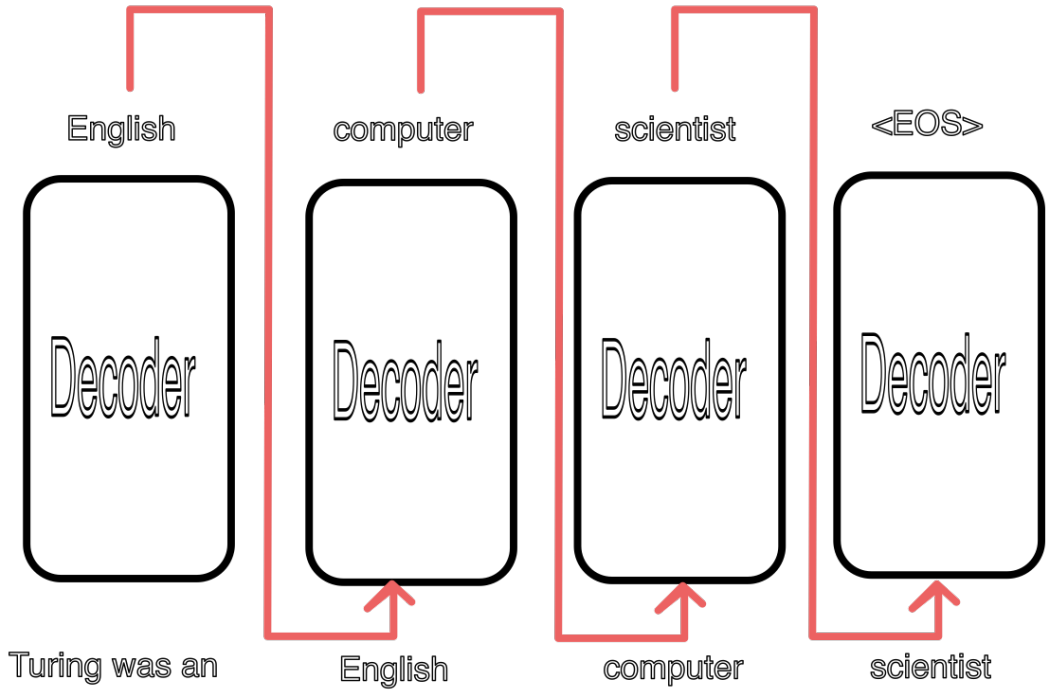
An execution includes **both the prompt** sent **and the response**. Language: English

Tokens per execution	Words per execution	Price for 1 execution	OpenAI price for 216000 executions	Using GPT for Work (Sheets or Excel) Including OpenAI cost With your own API key
10	8	~$0.00020	~$43.20	~$86.40
20	15	~$0.00040	~$86.40	~$172.80
50	38	~$0.00100	~$216.00	~$432.00
100	75	~$0.00200	~$432.00	~$864.00
200	150	~$0.00400	~$864.00	~$1728.00
500	375	~$0.01000	~$2160.00	~$4320.00
1000	750	~$0.02000	~$4320.00	~$8640.00
2000	1500	~$0.04000	~$8640.00	~$17280.00
4000	3000	~$0.08000	~$17280.00	~$34560.00

[Understanding the cost of Large Language Models]

The two-stage process

- Prefilling stage
  - Process all the input tokens at once
  - Compute- intensive
- Decoding stage
  - Generate one token at a time
  - Memory-intensive
- Process these two stage on the same GPU



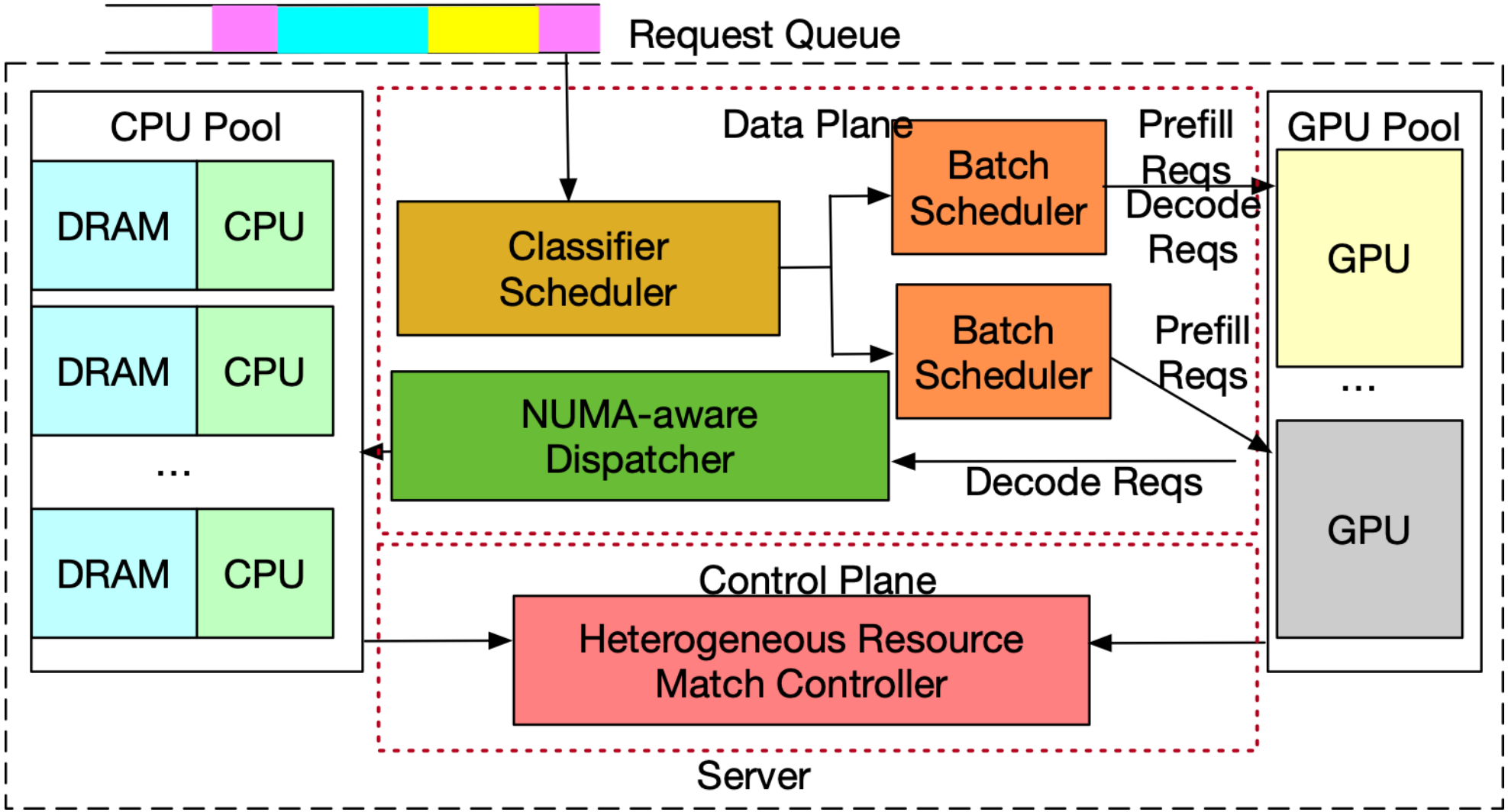
Key observations

- Current LLM inference systems often integrate the prefilling and decoding stages on a single GPU, which can lead to two types of interference.
  - The interference bewteen the prefilling and decoding
  - The interference within the prefilling
- CPU resource is more affordable than the expensive GPU resource

Our goal

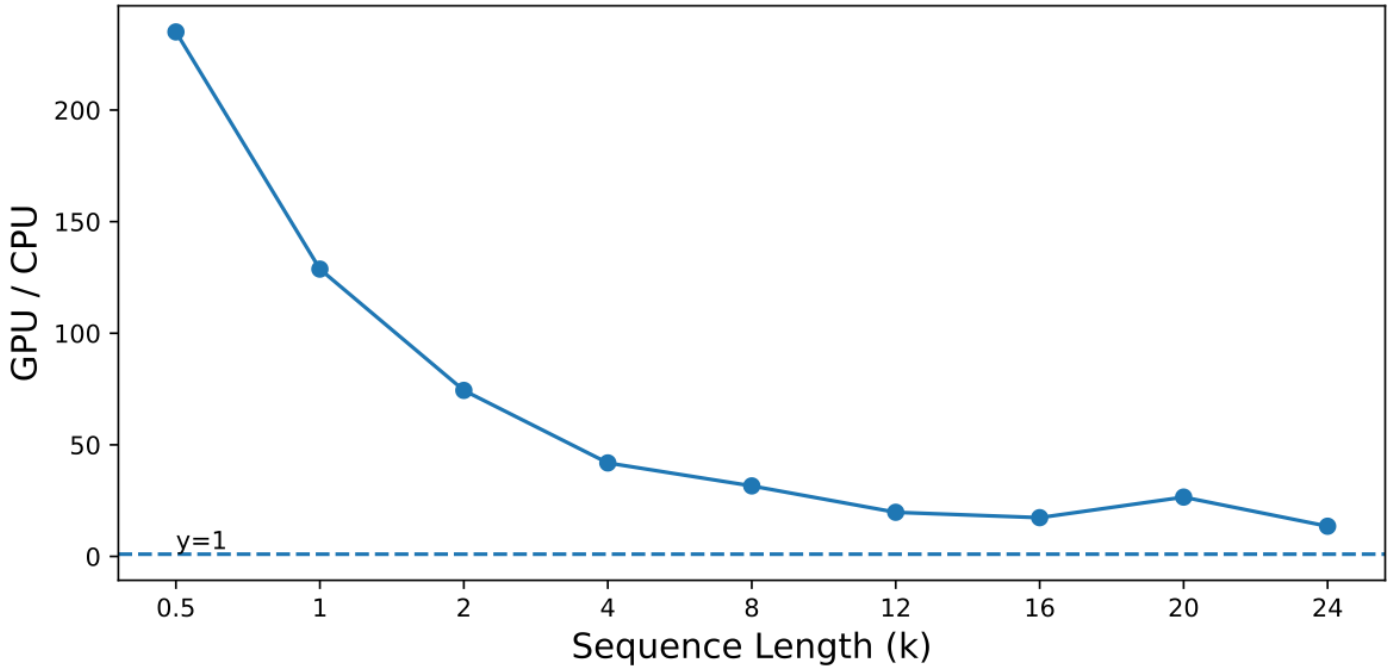
- We concentrate on mitigating the interference among the tasks of Large Language Model (LLM) services by asynchronously offloading the computational requests during the decoding stages to the CPU.

Architecture overview



Challenge 1 How many CPUs are required to match the token generate speed on GPUs?

- The resource allocation between CPU and GPU is influenced by the length of requests:
  - 512 tokens: GPU/CPU = 235
  - 24k tokens: GPU/CPU = 13.5
- User's requests come randomly



Solution 1 Heterogeneous Resource Match Controller

- Goals:
  - Ensure adequate CPU resources for decoding during periods of high workload burstiness .
  - Preventing excessive CPU idleness during times of low demand
- Solution:
  - An M/M/c queue : $\rho = \frac{\lambda}{c\mu}$ (1)
  - Poisson distribution
  - λ: mean inter-arrival times of batches completing the prefill process by the GPU
  - $\frac{1}{\mu}$: latency of the decoding process
  - c: the number of CPUs

Solution 1 Heterogeneous Resource Match Controller

- Solution:
  - $$E(W) = \Pi_w \times \frac{1}{1-\rho}\frac{1}{c\mu} \qquad (2)$$
- Three steps to allocate the CPU:
  - Profile the GPUs and the workloads to figure out the arrival rate λ
  - Profile the CPUs and the requests requiring decoding to obtain the mean decoding latency
  - Set an atcceptable waiting time threshold

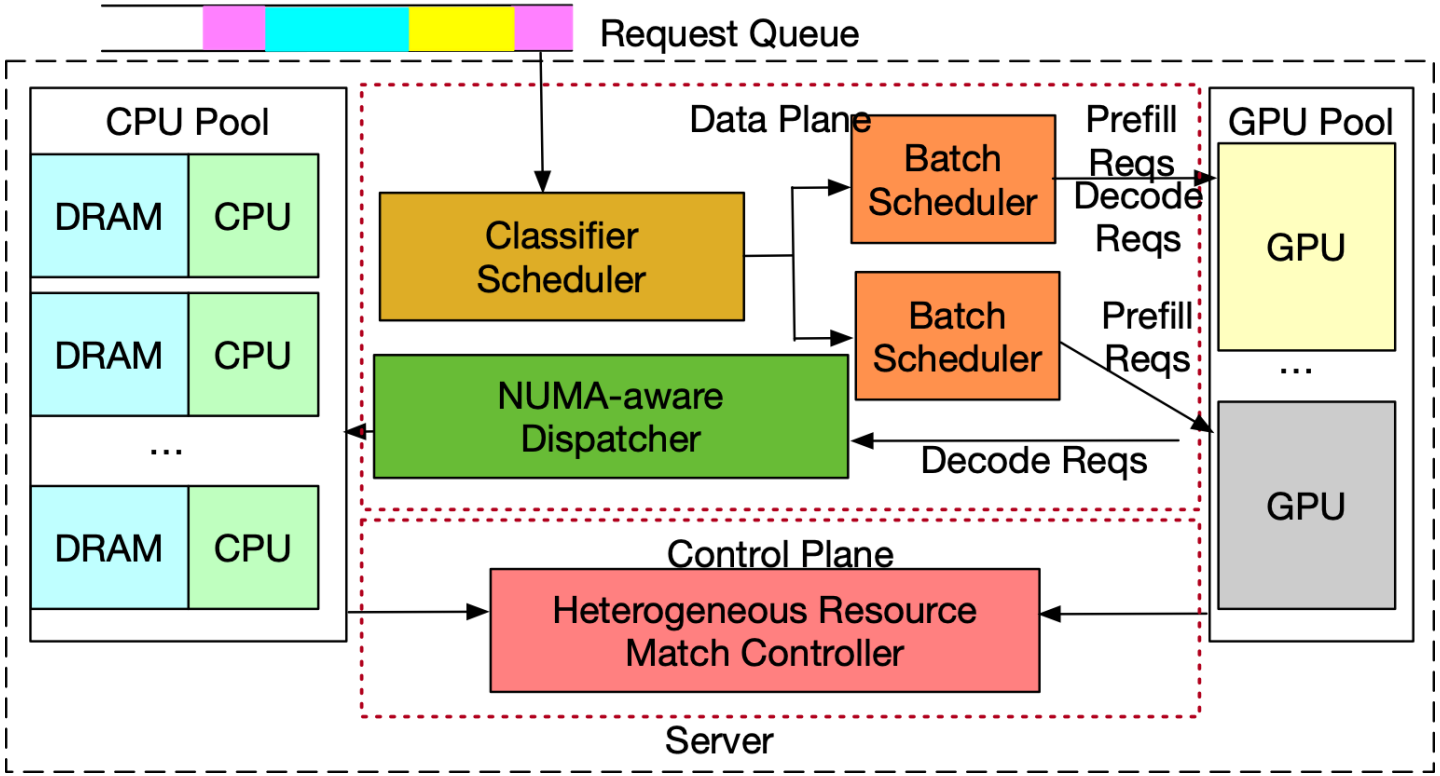
Challenge 2: How to prevent mutual interference among requests with different lengths during prefilling?

- Within the prefilling stage, the varying lengths of requests within the same batch can also cause interference.

- Our preliminary experiment shows that compared to using batches with similar prompt lengths, forming batches with random request lengths leads to a decrease in prefilling throughput: from 2.57 q/s(question per second) to 2.0 q/s.

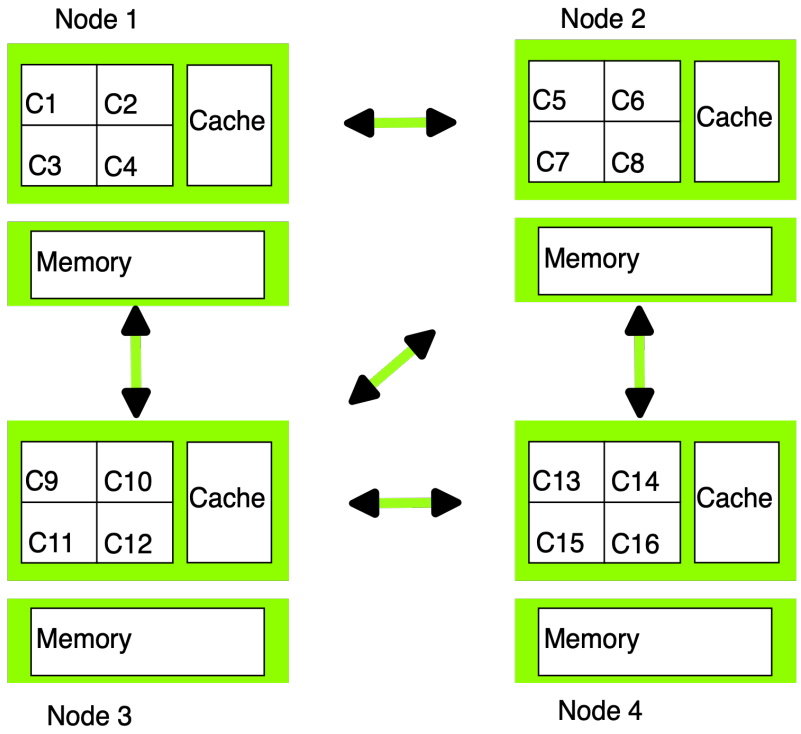
Solution 2. Hierarchical GPU Scheduler

- Core idea: groups requests of similar lengths into batches



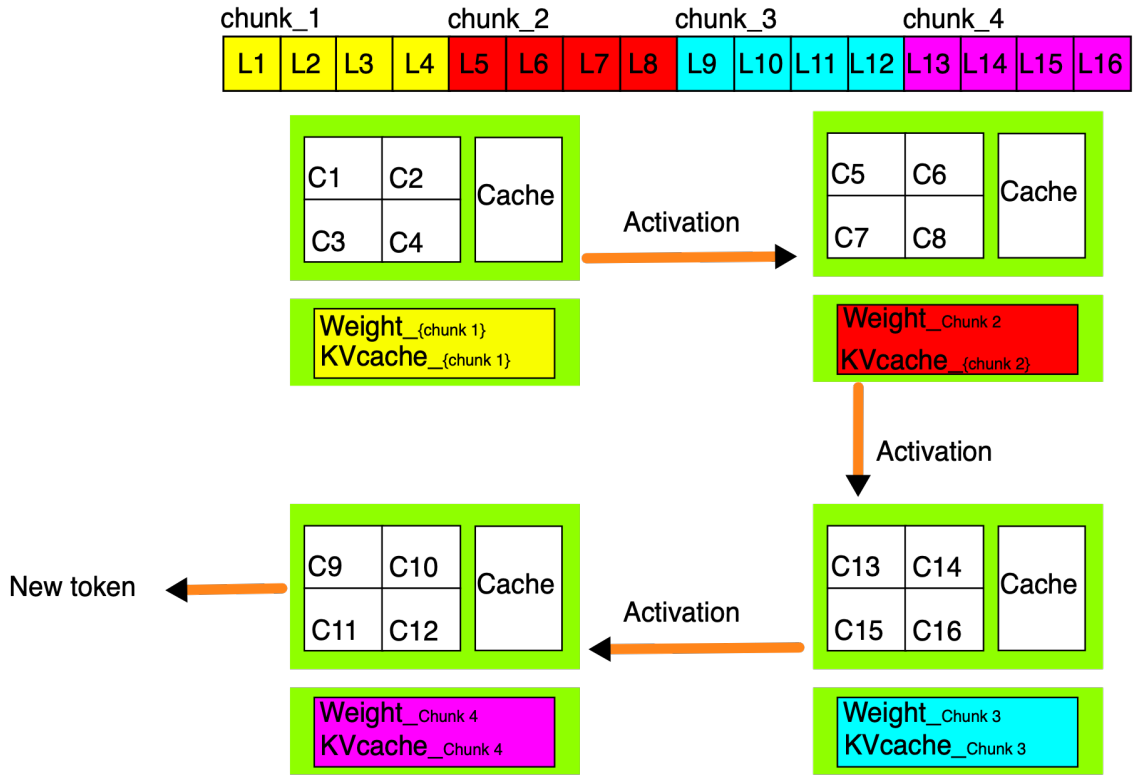
Challenge 3: How to reduce cross-node data access overhead under the NUMA architecture?

- The performance of NUMA can degrade significantly if numerous memory transactions occur remotely from another socket.
- Remote access takes ~30% longer than a local access, while on older hardware, it could take up to 7 times longer.



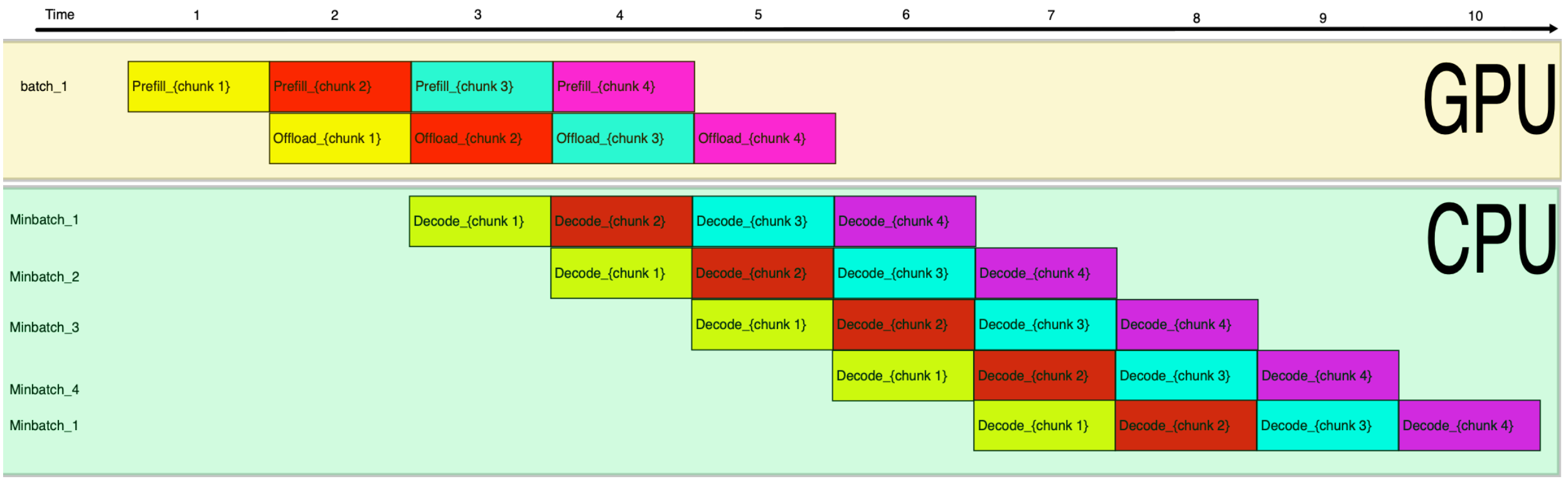
Solution 3: NUMA-Aware Dispatcher

- The core idea is to couple computation and data placement on the same socket.
- placing a higher value on the placement of model weights than on that of the KV cache



Solution 3: NUMA-Aware Dispatcher

- The GPU prefill a batch of request
- Devide a batch into sevral min-batch
- Select a min-batch of requests and excute them using a pipeline method



Discussion

- Adapting to Faster CPU
  - Our Setup
    - GPU: A100
    - CPU: Intel(R) Xeon(R) Gold 6148 CPU
- Using RL for Resource Management

Conclusion

- hLLM is designed to offload the decoding stages to the CPU, effectively reducing the interference that can occur between the prefilling and decoding phases.
- hLLM intelligently allocates GPU and CPU resources to optimize the distribution of computational capabilities across different devices.
- hLLM boosts GPU prefill throughput by employing a task scheduling strategy that mitigates internal interference.
- hLLM designs a NUMA-aware dispatcher to avoid performance degradation.

Thank you!