USENIX
ATC '25

Weaver: Efficient Multi-LLM Serving with Attention Offloading

Shiwei Gao, Qing Wang, Shaoxun Zeng, Youyou Lu and Jiwu Shu

Presenter: Hao Guo

Tsinghua University



Outline

- Background & Motivation
- Key Idea and Challenge
- System Design
- Evaluation
- Conclusion

LLM Usage Today



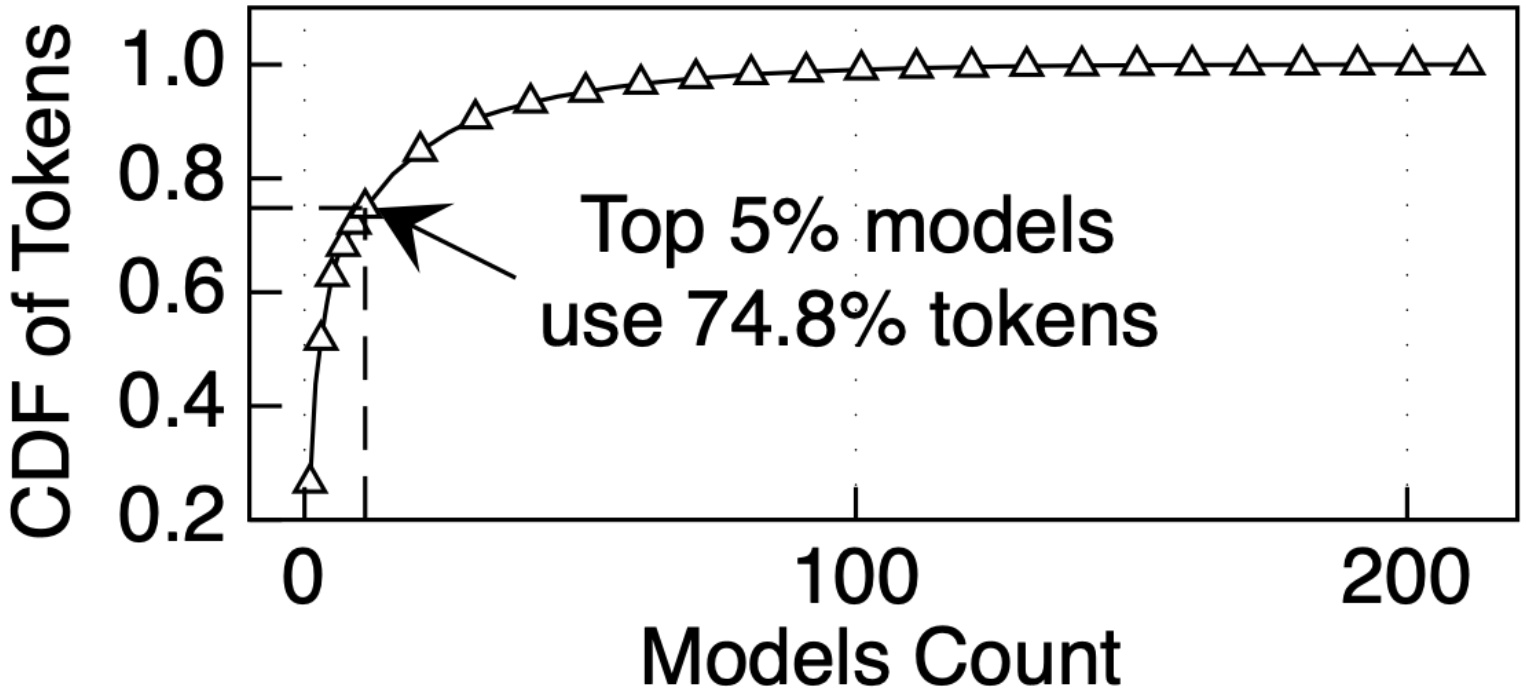


Serving Multiple LLMs Concurrently is a Trend!

Model-as-a-Service platforms serving **hundreds of models**!

Single large company(e.g., OpenAI) serving **tens of models**!

Multi-LLM Serving Characteristic

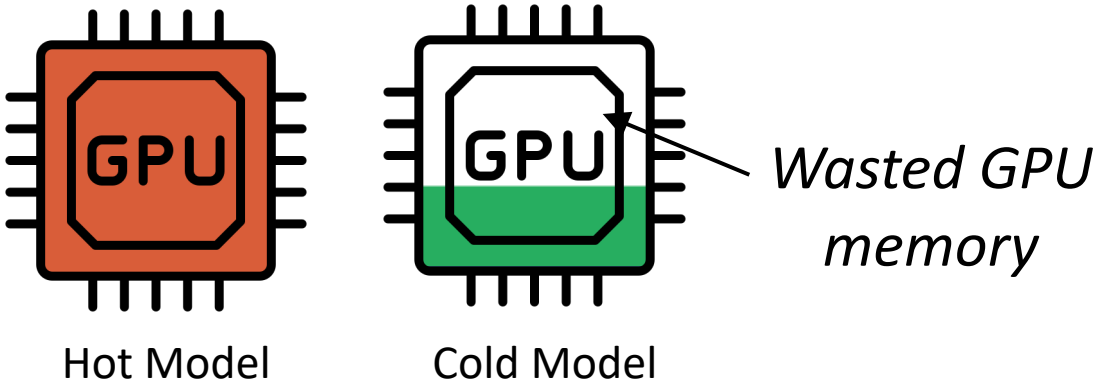


Request loads are *skewed* between models

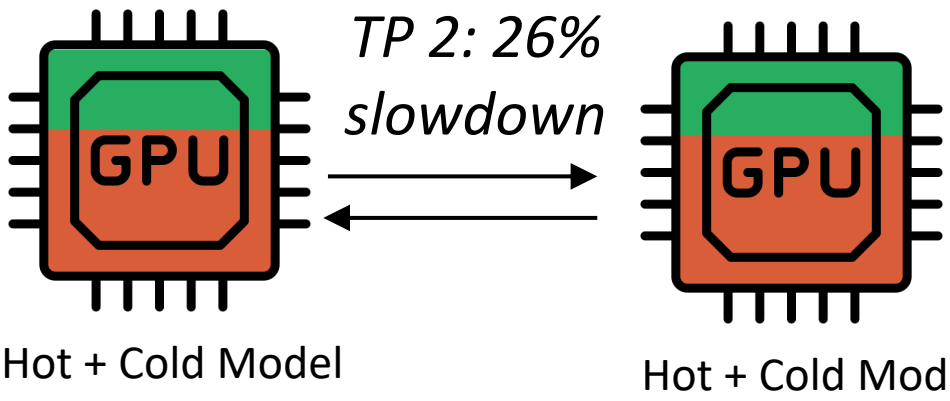
[1] OpenRouter, https://openrouter.ai/

Existing Methods for Multi-LLM Serving

#1 Dedicated Serving



#2 Model Parallelism



☹ Underutilize GPU memory

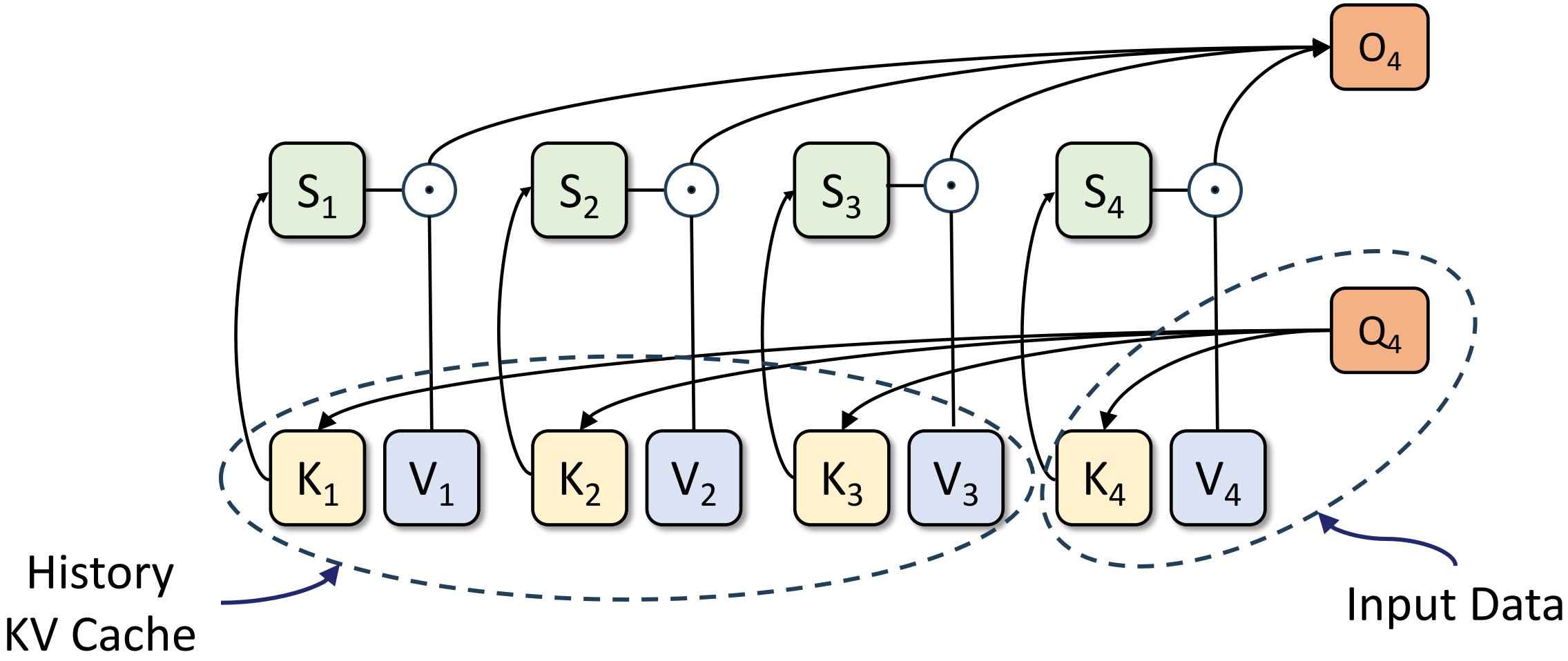
☹ Bring Communication overhead

Design Goal: maximize multi-LLM GPU memory utilization, *without* sacrificing performance.

Outline

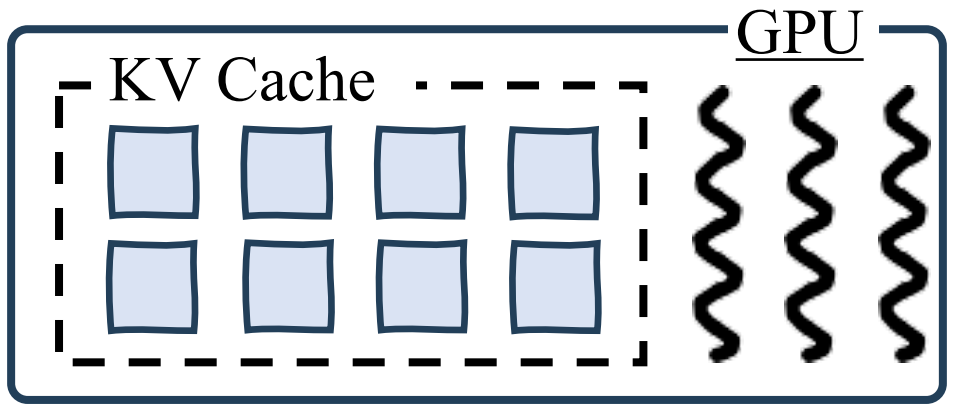
- Background & Motivation
- Opportunity and challenges
- System Design
- Evaluation
- Conclusion

Attention Computing



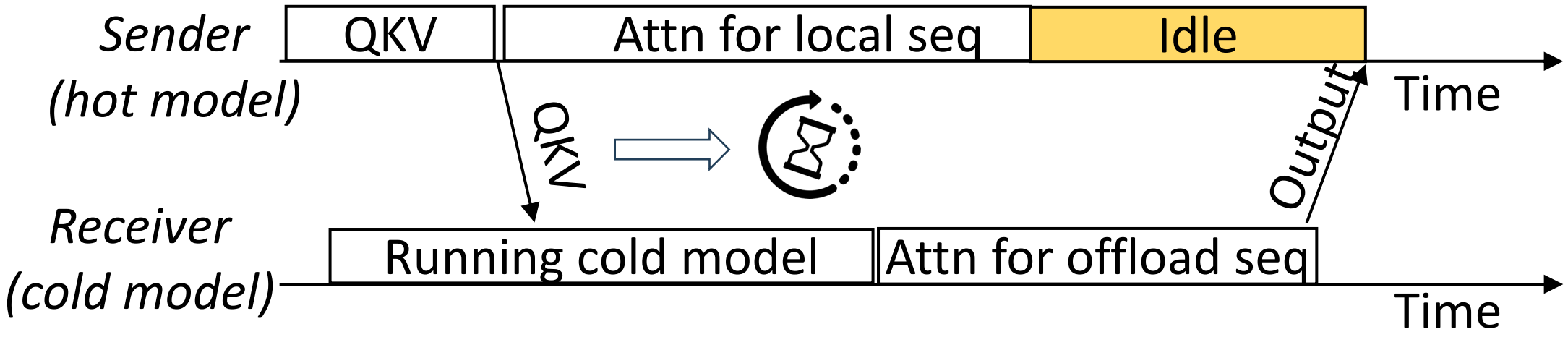
Only QKVO vector of the new token is needed

Opportunity: Attention Offloading



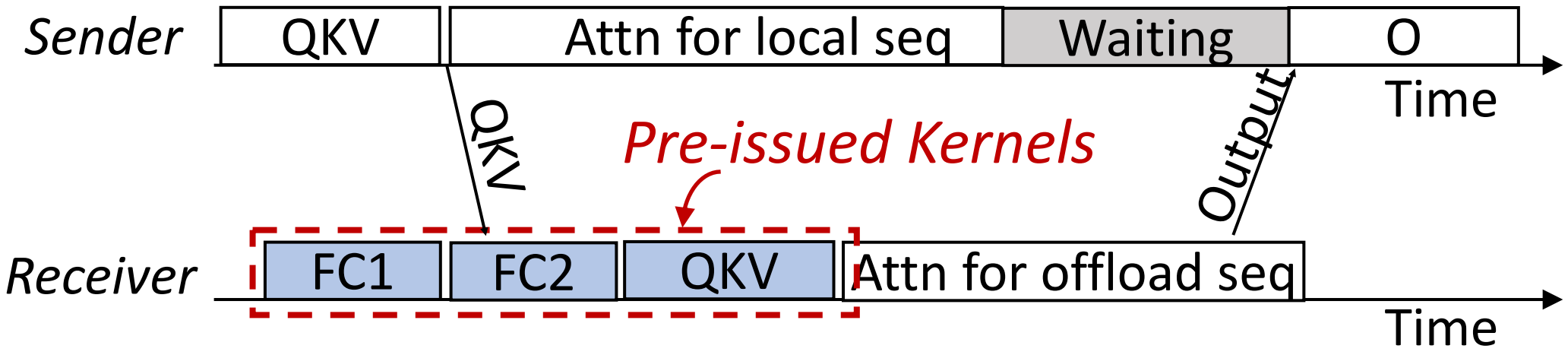
Attention offloading can balance the memory utilization with small communication overhead

What's the new problem?



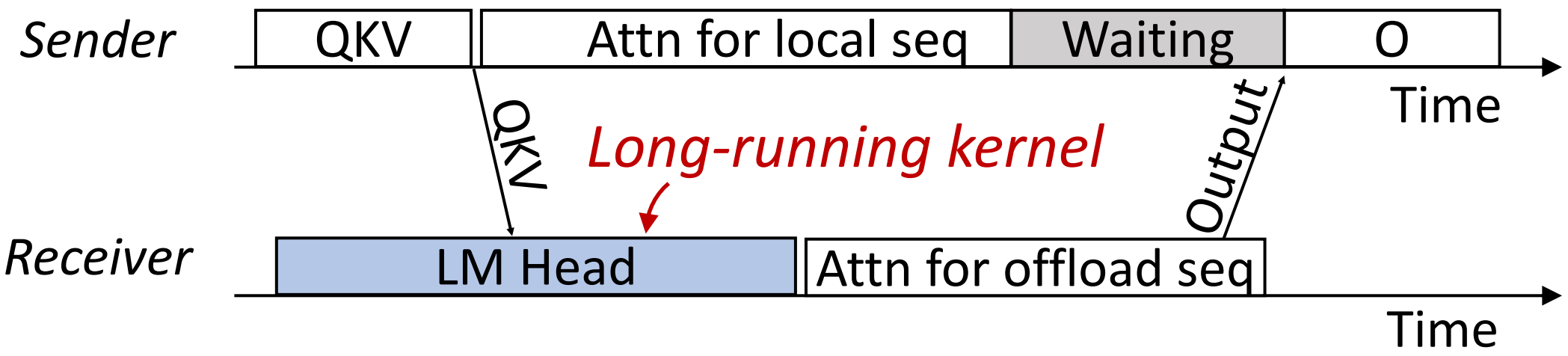
The offloading target is ***not dedicated***
Running cold model ***blocks*** offloaded attention

Challenge #1



The offloaded attention is blocked by many *pre-issued kernels*

Challenge #2



The offloaded task is blocked by a *single long-running kernel*

Outline

- Background & Motivation
- Key Idea and Challenge
- System Design
- Evaluation
- Conclusion

Our Approach

Challenges

C1:Blocked by pre-issued kernels

C2: Blocked by long kernel

Solutions

GPU-driven Control Flow

Insert polling after each kernel

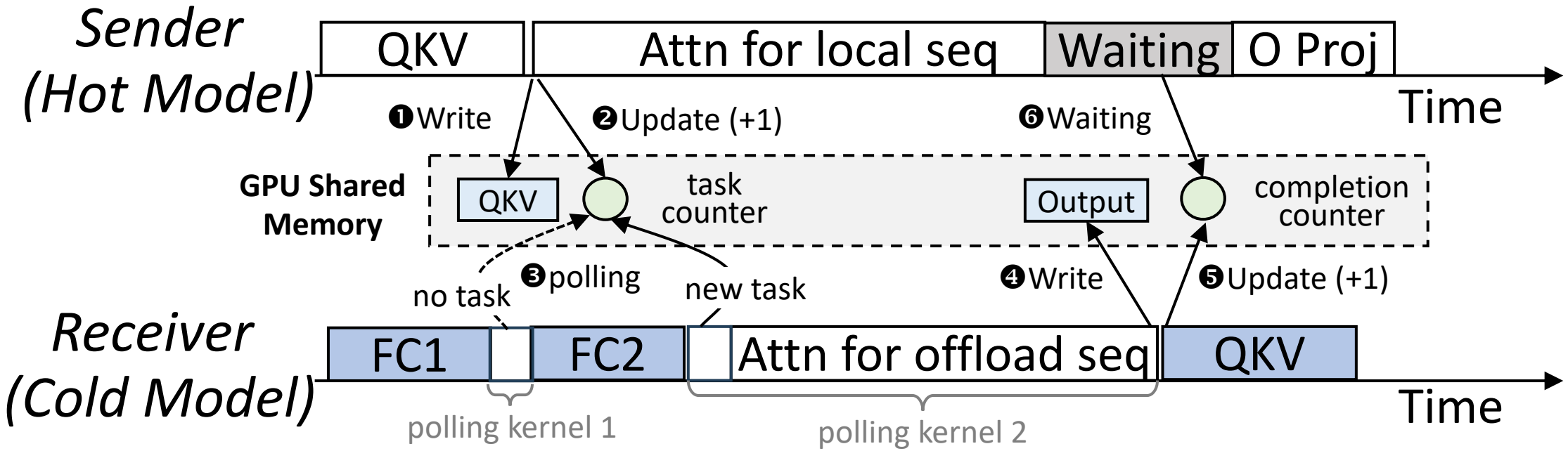
Operator Splitting

Break a large kernel into pieces

Outcome

✅ Achieve non-blocking attention offloading

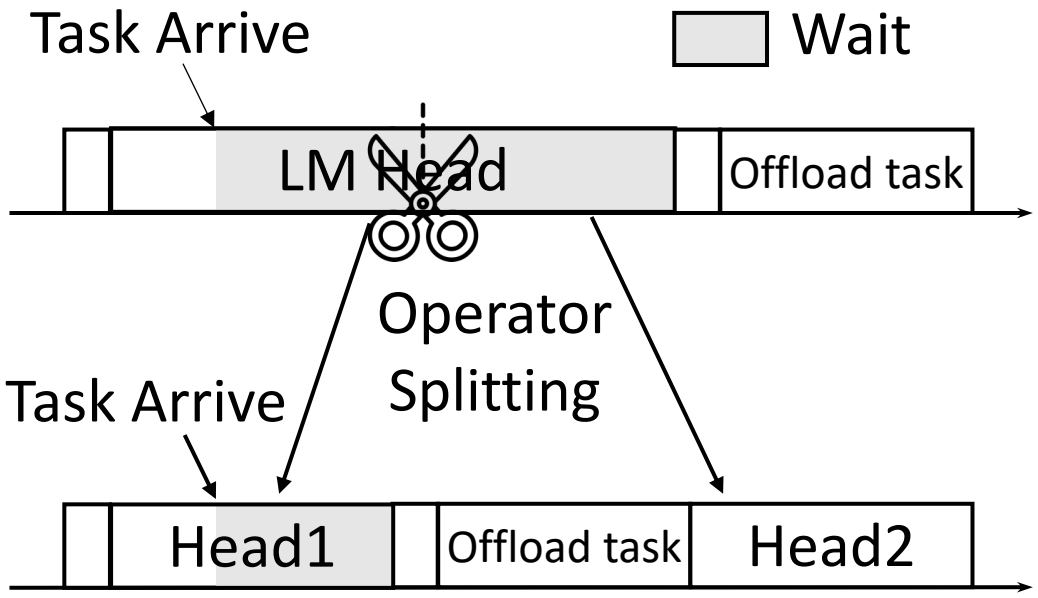
GPU Driven Control Flow



Offloaded computation is now blocked by only one kernel

Operator Splitting

Simple Idea: Breaking large kernels



Mathematic Observation

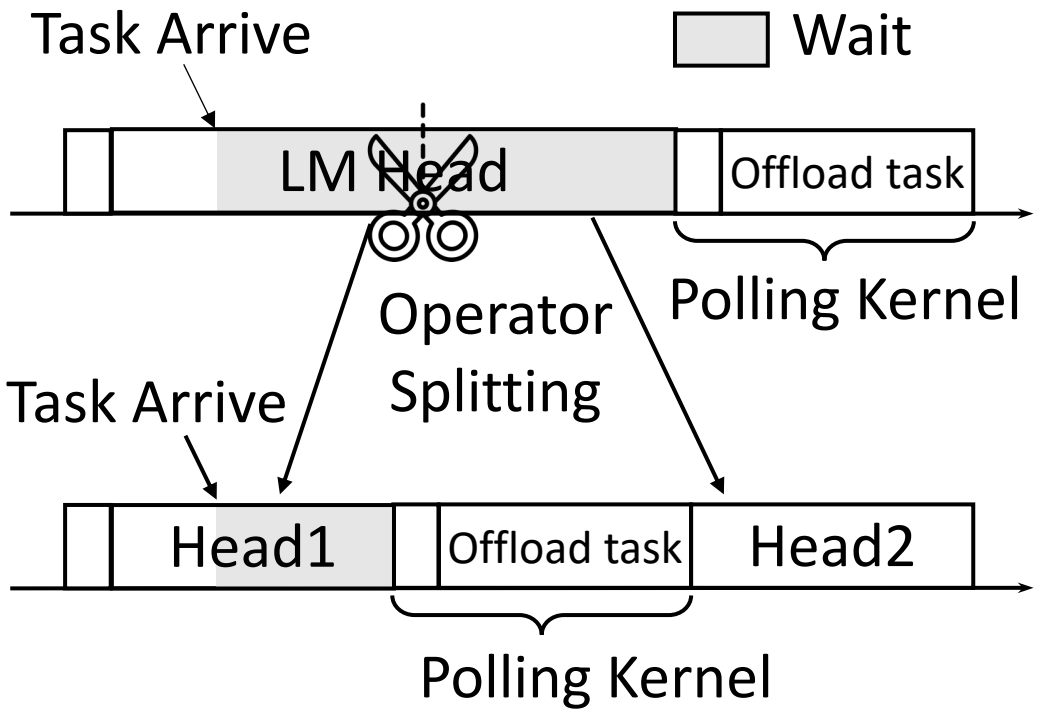
Each Kernel ***quadratically*** contributes to the waiting time



① Operator priority queue: sort by the operator's running time

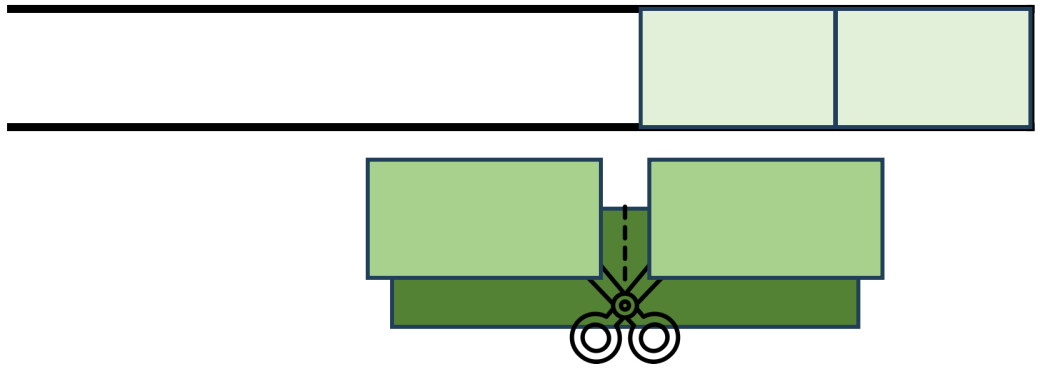
Operator Splitting

Simple Idea: Breaking large kernels



Mathematic Observation

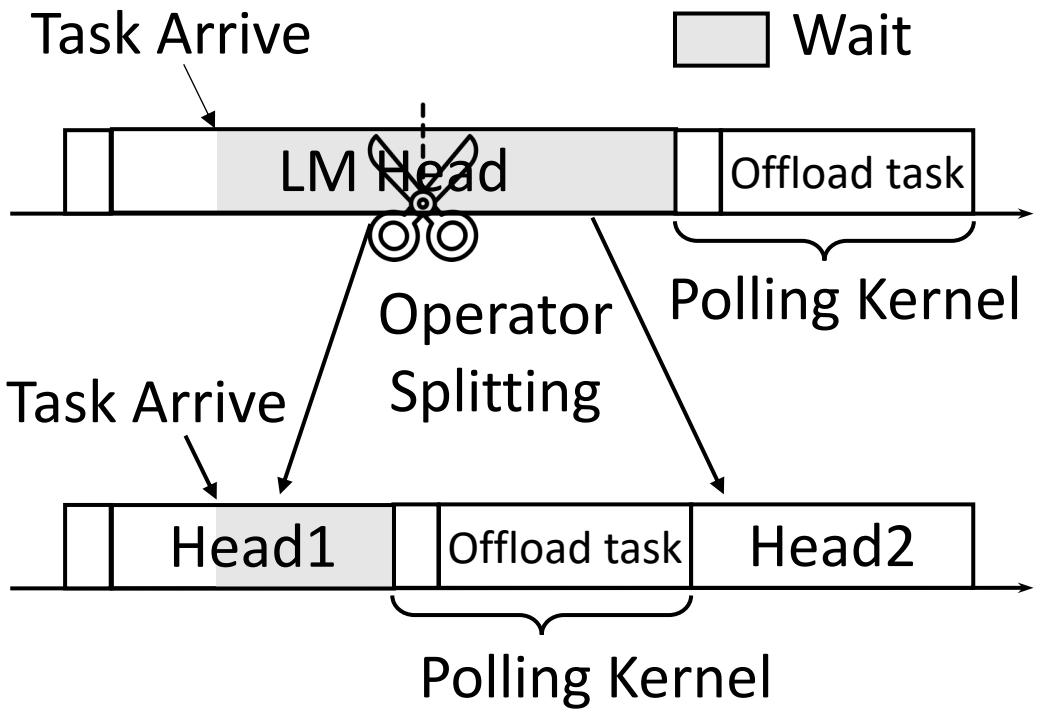
Each Kernel ***quadratically*** contributes to the waiting time



② Split the biggest operator into two halves and insert back

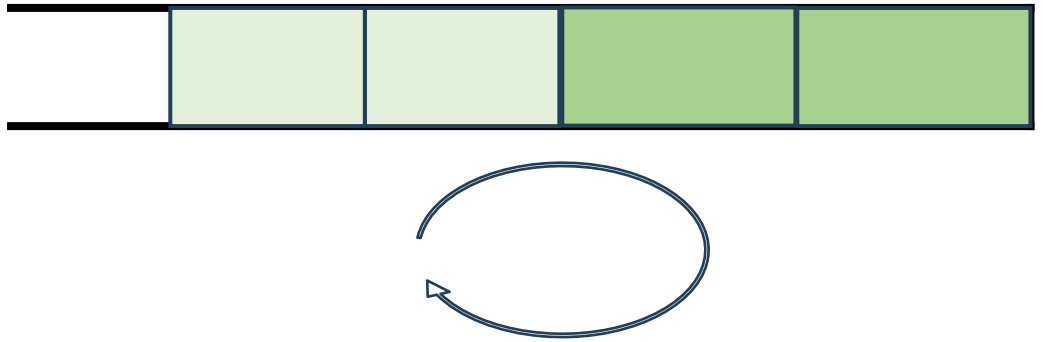
Operator Splitting

Simple Idea: Breaking large kernels



Mathematic Observation

Each Kernel ***quadratically*** contributes to the waiting time



③ Reinsert into the queue. Repeat until the waiting time < threshold

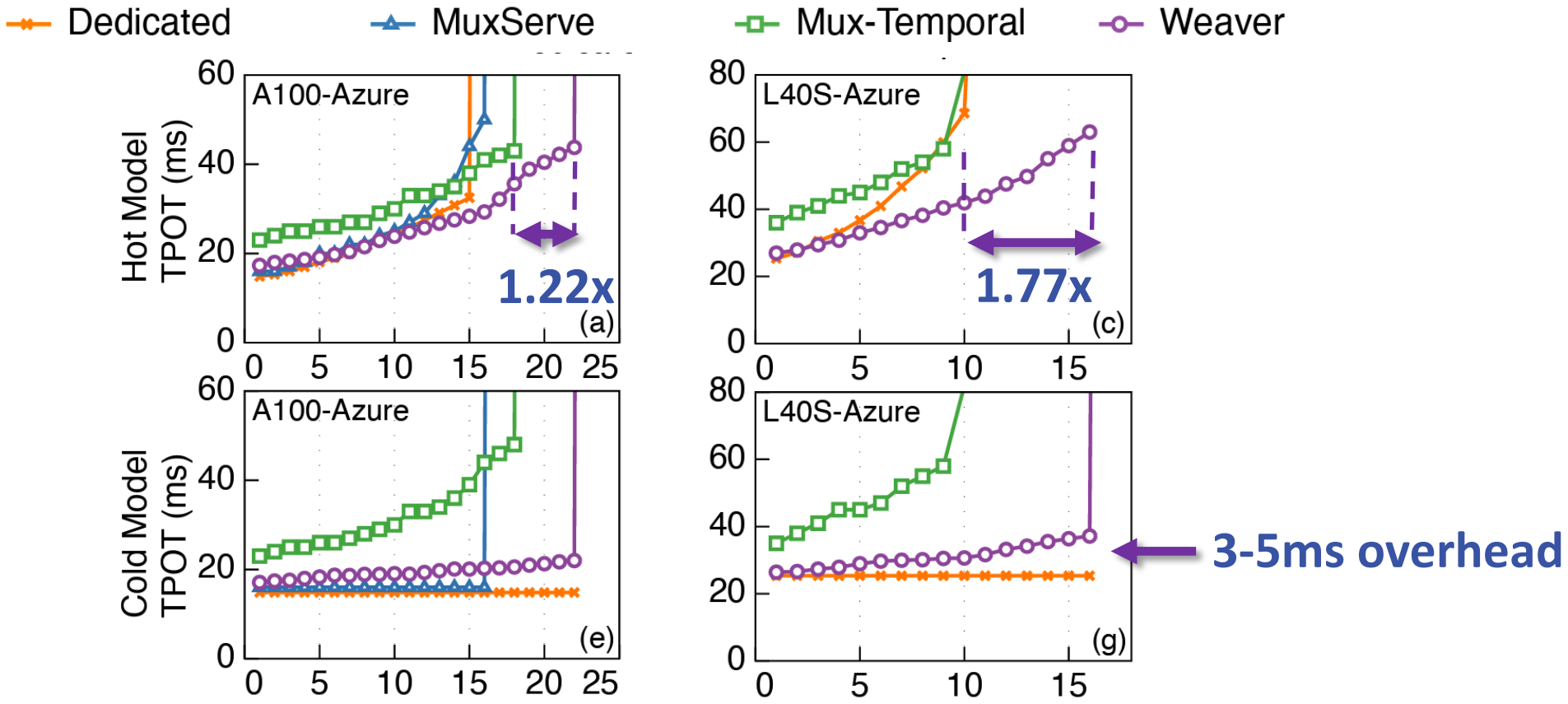
Outline

- Background & Motivation
- Key Idea and Challenge
- System Design
- **Evaluation**
- Conclusion

Evaluation

- Baselines:
  - Dedicated Serving$_{[vLLM, v0.6.0]}$
  - MuxServe (With MPS)
  - MuxServe-Temporal
- Testbed:
  - A100-40GB (connected with NVLink)
  - L40S (connected with PCIe)
- Workloads:
  - BurstGPT and Azure-Conversation

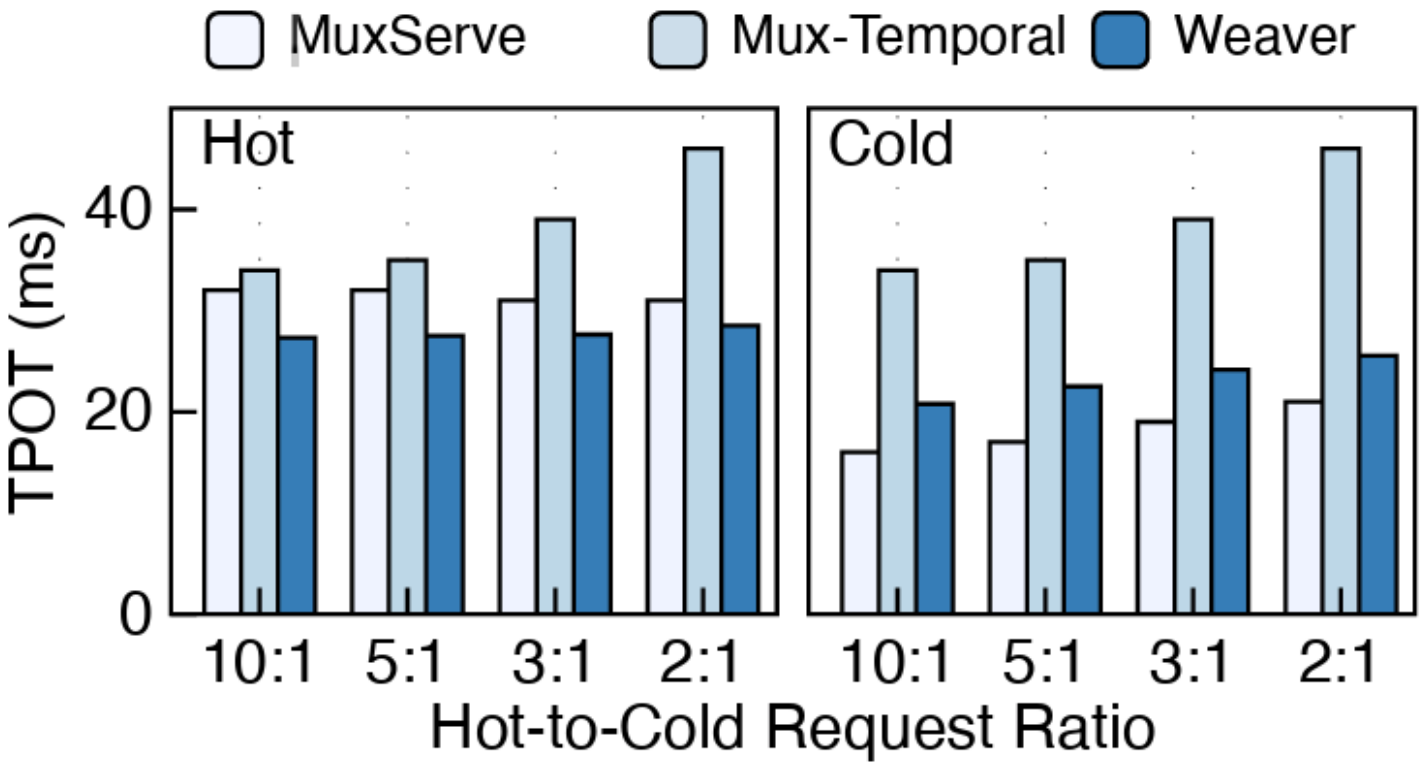
Overall Performance



Weaver

- Throughput: achieve **1.22x** higher on A100 and **1.77x** higher on L40S
- TPOT: on average 3- 5ms overhead to the cold model

Sensitivity Analysis



Weaver

- Show benefits (*15% lower TPOT*) under different hot-to-cold request ratios

Outline

- Background & Motivation
- Key Idea and Challenge
- System Design
- Evaluation
- Conclusion

Thank you!

Weaver: efficient multi-LLM serving system with attention offloading

- **Workload: skewed** between models (hot and cold)
- **Opportunity:** offloading attention from hot to cold instances
- Two key techniques:
  - **GPU Driven Control Flow**: polling kernel + shared mem comm
  - **Operator Splitting:** break a large kernel into pieces
- **Higher throughput,** small overhead on the cold model**.**

Contact: gsw23@mails.tsinghua.edu.cn



image: Flaticon.com ☺