NanoFlow: Towards Optimal Large Language Model Serving Throughput

Kan Zhu, *University of Washington;* Yufei Gao, *Tsinghua University and University of Washington;* Yilong Zhao, *University of Washington and University of California, Berkeley;* Liangyu Zhao, *University of Washington;* Gefei Zuo, *University of Michigan;* Yile Gu and Dedong Xie, *University of Washington;* Tian Tang and Qinyu Xu, *Tsinghua University and University of Washington;* Zihao Ye, Keisuke Kamahori, and Chien-Yu Lin, *University of Washington;* Ziren Wang, *Tsinghua University and University of Washington;* Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci, *University of Washington*

OSDI' 25

Presented by Yinhe Chen, Dongqi Tian

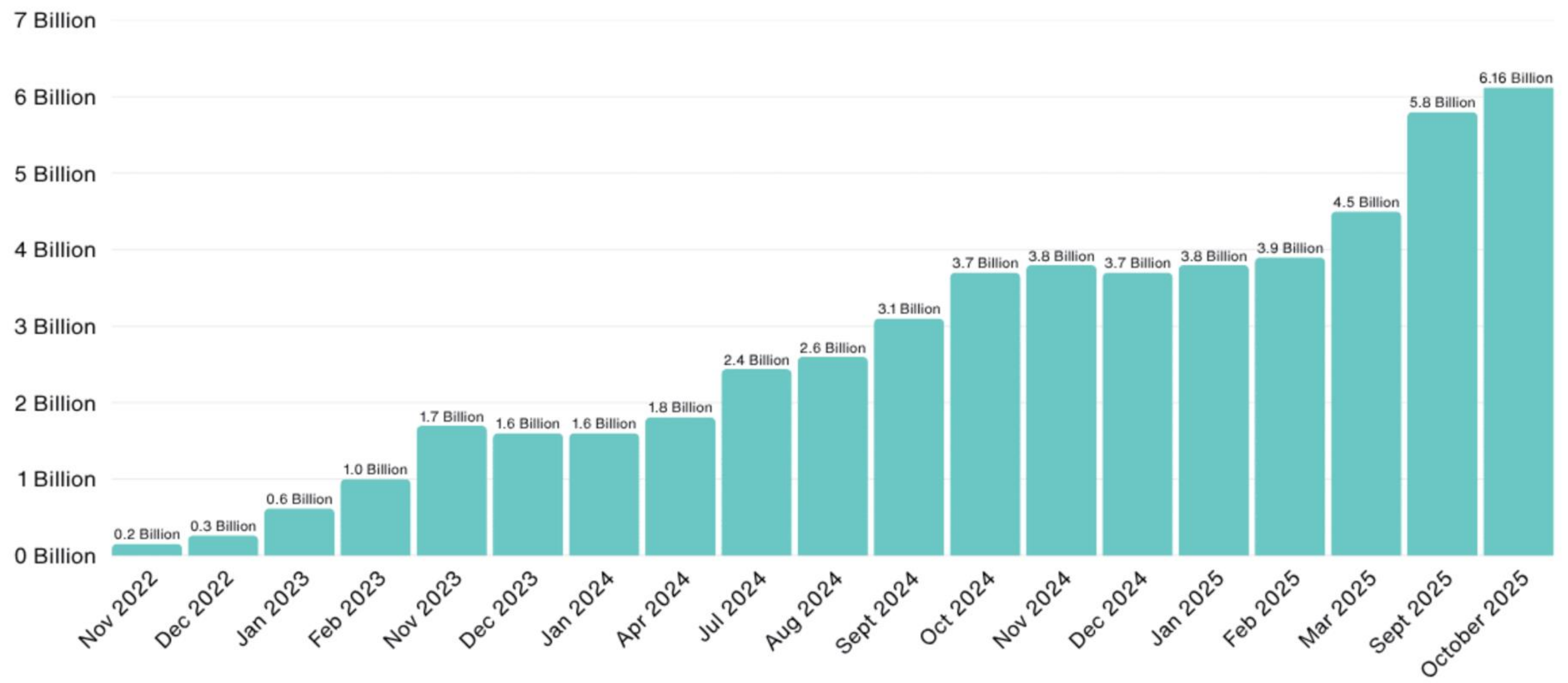
USTC, CHINA ADSLAB 先进数据系统实验室

Outline

- Background
    - LLM inference workflow and OPs
- Analysis
- Design & Implementation
- Evaluation

Background

- LLM apps are getting popular
  - ◆ a high throughput serving system is essential

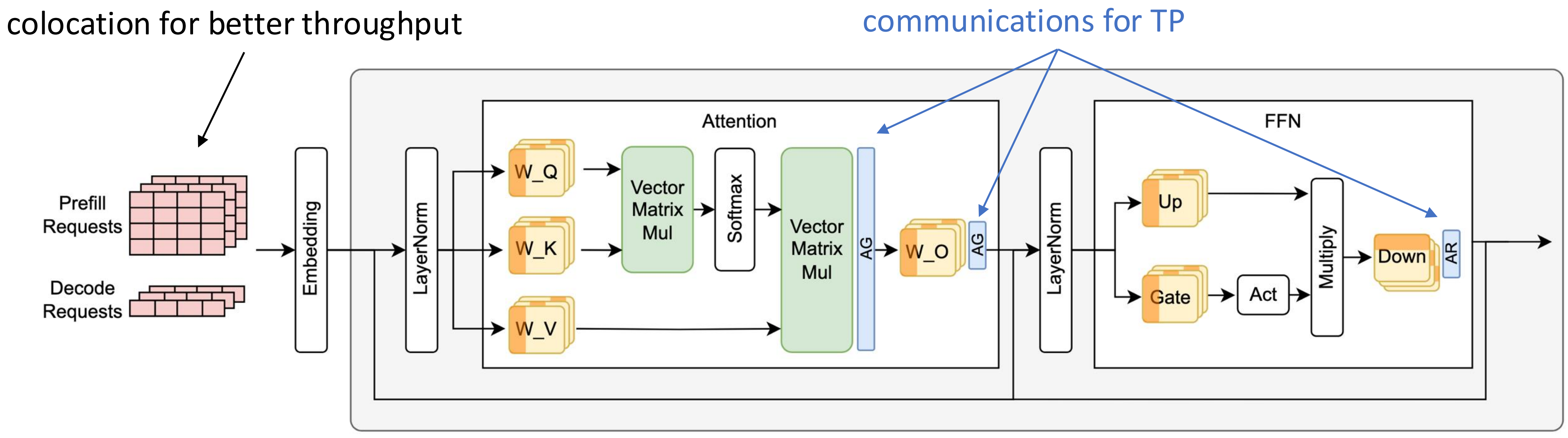


ChatGPT traffic by month [1]

[1] https://www.demandsage.com/chatgpt-statistics/

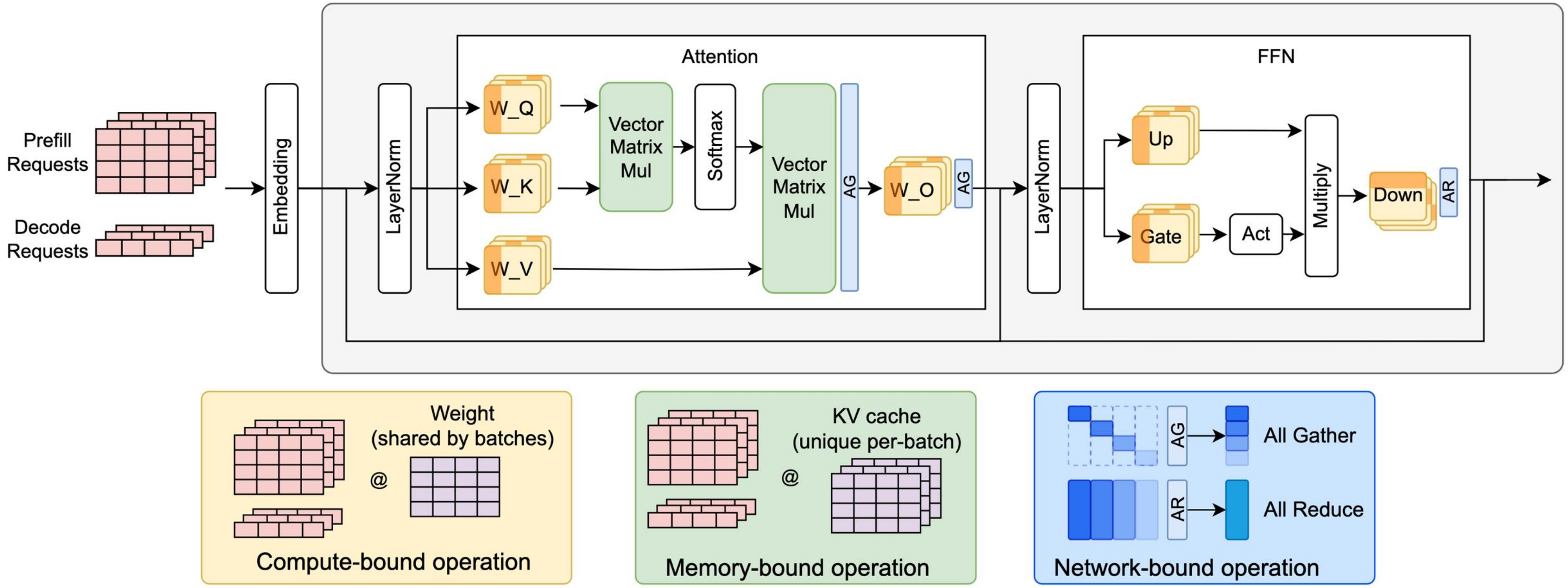
Background: LLM Inference

- LLM inference workflow



Background: LLM Inference

- OPs categories
  - compute-bound, memory-bound and network-bound



P & D requests share the same weight

each request has unique KV cache

Outline

Modeling LLM Inference

Notation	Meaning
MemSize	GPU memory capacity (GB)
MemBW	GPU memory bandwidth (GB/s)

- Memory time in each iteration
  - batch size should utilize all memory
    - the largest possible batch size provides highest throughput
  - **entire memory** is loaded to GPU cache
  - thus,

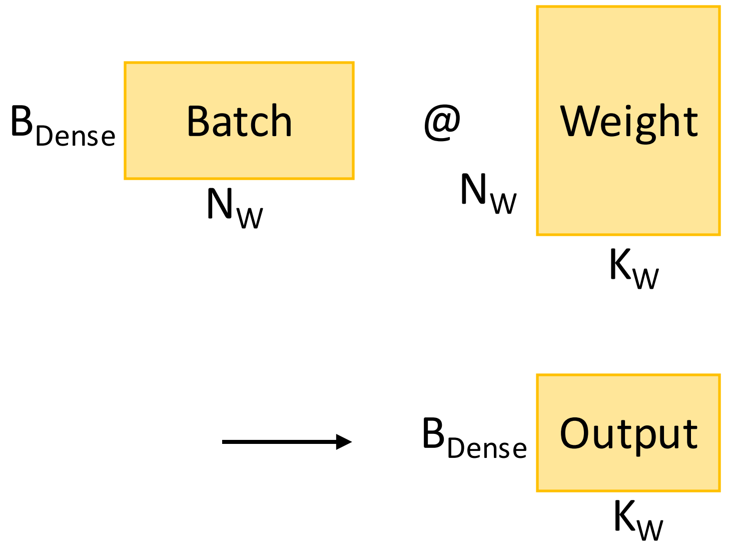
$$T_{mem} = \frac{MemSize}{MemBW}$$

Modeling LLM Inference

Notation	Meaning
B_{Dense}	Batched tokens for dense Ops
N_w, K_w	Dims of weight matrices
P_{Model}	Number of model parameters

- Compute time in each iteration
  - mainly focus GEMM (most compute intensive)
  - computations for each GEMM is $2B_{Dense}N_wK_w$
  - total GEMM computations is $2B_{Dense}\sum N_wK_w$
  - approximate $\sum N_wK_w \approx P_{Model}$
  - thus,

$$T_{Compute} \approx \frac{2B_{Dense} \cdot P_{Model}}{Compute}$$



GEMM example

Modeling LLM Inference

Notation	Meaning
B_{Dense}	Batched tokens for dense Ops
N_w, K_w	Dims of weight matrices
P_{Model}	Number of model parameters

A100 Spec.	Value
MemSize	80 GB = 8 x 10^10 B
MemBW	2,000 GB/s = 2 x 10^12 B/s
Compute	312 TFLOP/s = 3.12 x 10^14 FLOP/s
Compute / MemBW	156 FLOPs / B

- Mem vs Comp

156 FLOPs/B 8 x 8 x 10^10 B

$$T_R = \frac{T_{Mem}}{T_{Compute}} \approx \boxed{\frac{Compute}{MemBW}} \frac{MemSize}{P_{model}} \frac{1}{2B_{dense}}$$

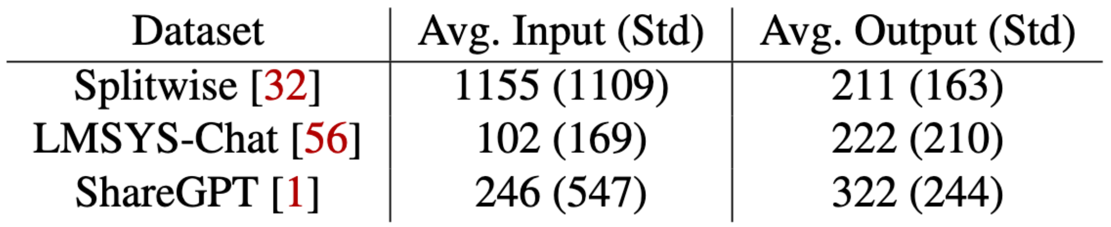
< 156 x 10 / 4096 < 1

70 x 10^9 2 x 2048

- Example
  - Llama-2 70B on 8xA100 80G
  - B_{dense} is 2048 (1024 decoding requests + 1024 prefill tokens)
  - T_R < 1 indicates **compute-bound**

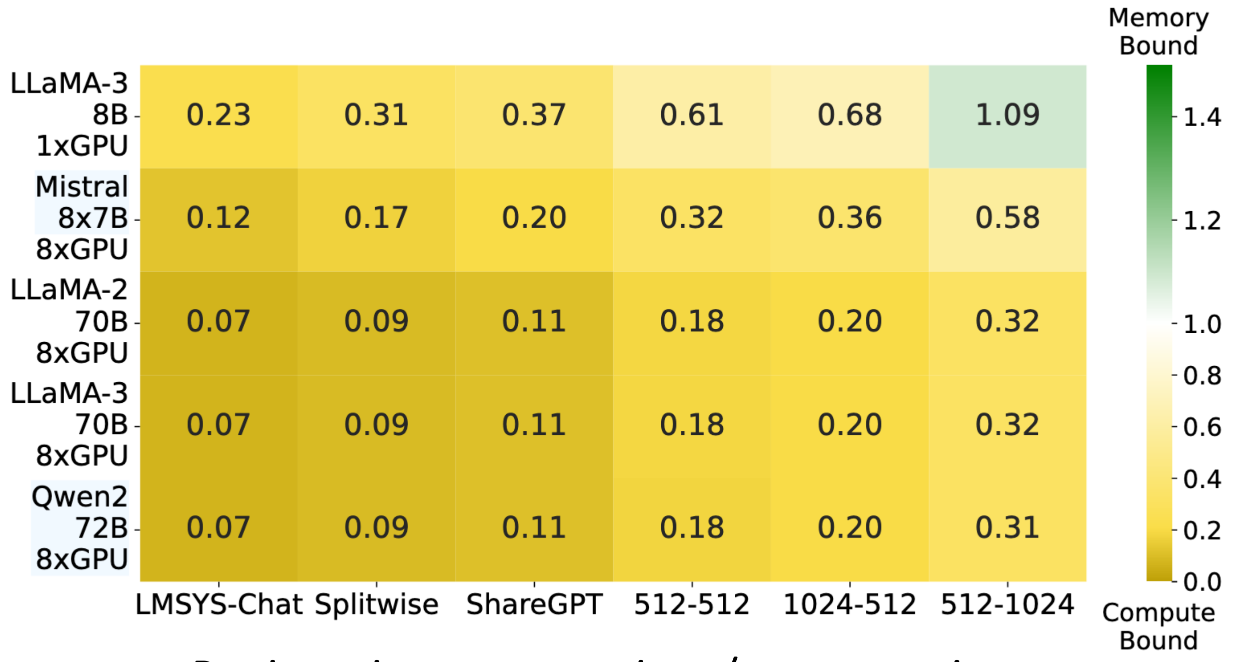
LLM Inference is Compute-bound

- Conditions
  - PD colocation
  - short requests
    - (P < 2K, D < 500)



Dataset	Avg. Input (Std)	Avg. Output (Std)
Splitwise [32]	1155 (1109)	211 (163)
LMSYS-Chat [56]	102 (169)	222 (210)
ShareGPT [1]	246 (547)	322 (244)

Request lengths in tested datasets.

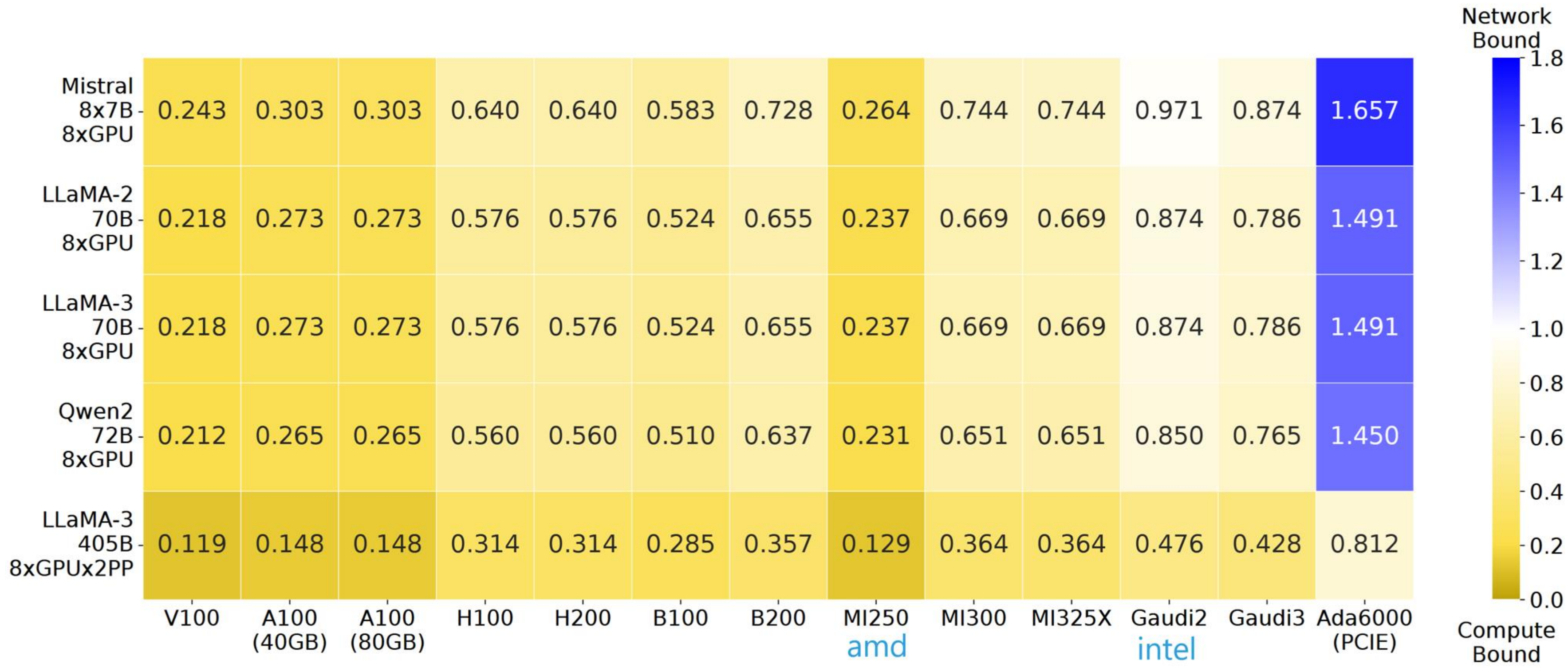


	LMSYS-Chat	Splitwise	ShareGPT	512-512	1024-512	512-1024
LLaMA-3 8B 1xGPU	0.23	0.31	0.37	0.61	0.68	1.09
Mistral 8x7B 8xGPU	0.12	0.17	0.20	0.32	0.36	0.58
LLaMA-2 70B 8xGPU	0.07	0.09	0.11	0.18	0.20	0.32
LLaMA-3 70B 8xGPU	0.07	0.09	0.11	0.18	0.20	0.32
Qwen2 72B 8xGPU	0.07	0.09	0.11	0.18	0.20	0.31

Per iteration memory time / compute time.
Yellow means compute bound.

LLM Inference is Compute-bound

- Network vs Compute
  - compute time > network time (except for PCIe)



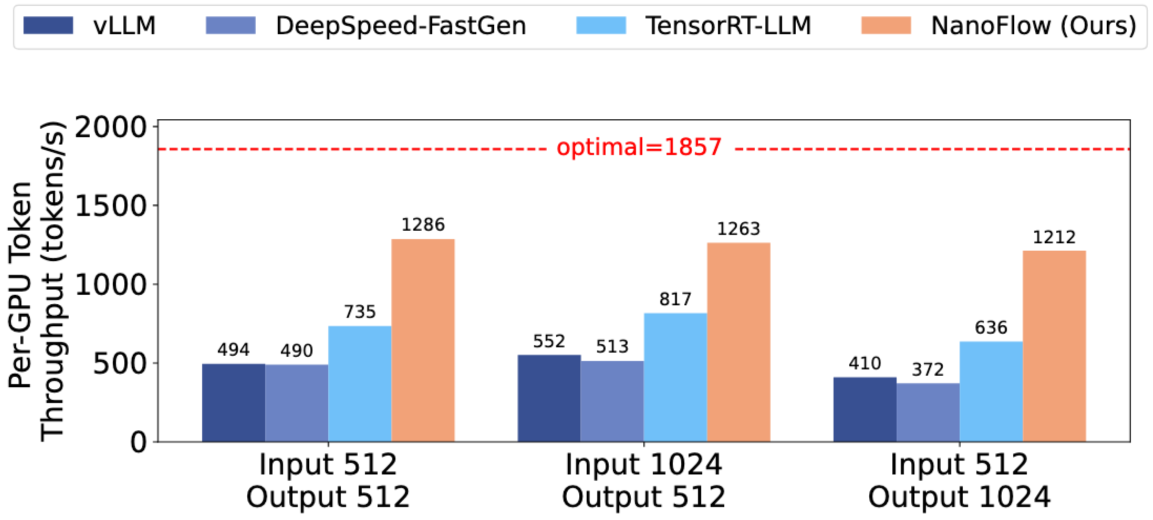
	V100	A100 (40GB)	A100 (80GB)	H100	H200	B100	B200	MI250	MI300	MI325X	Gaudi2	Gaudi3	Ada6000 (PCIE)
Mistral 8x7B 8xGPU	0.243	0.303	0.303	0.640	0.640	0.583	0.728	0.264	0.744	0.744	0.971	0.874	1.657
LLaMA-2 70B 8xGPU	0.218	0.273	0.273	0.576	0.576	0.524	0.655	0.237	0.669	0.669	0.874	0.786	1.491
LLaMA-3 70B 8xGPU	0.218	0.273	0.273	0.576	0.576	0.524	0.655	0.237	0.669	0.669	0.874	0.786	1.491
Qwen2 72B 8xGPU	0.212	0.265	0.265	0.560	0.560	0.510	0.637	0.231	0.651	0.651	0.850	0.765	1.450
LLaMA-3 405B 8xGPUx2PP	0.119	0.148	0.148	0.314	0.314	0.285	0.357	0.129	0.364	0.364	0.476	0.428	0.812

Per iteration network time / compute time.
Yellow means compute bound.

Optimal Throughput

$$\text{Throughput}_{\text{optimal}} = \frac{B_{Dense}}{T_{Compute}} = \frac{Compute}{2P_{Model}}$$

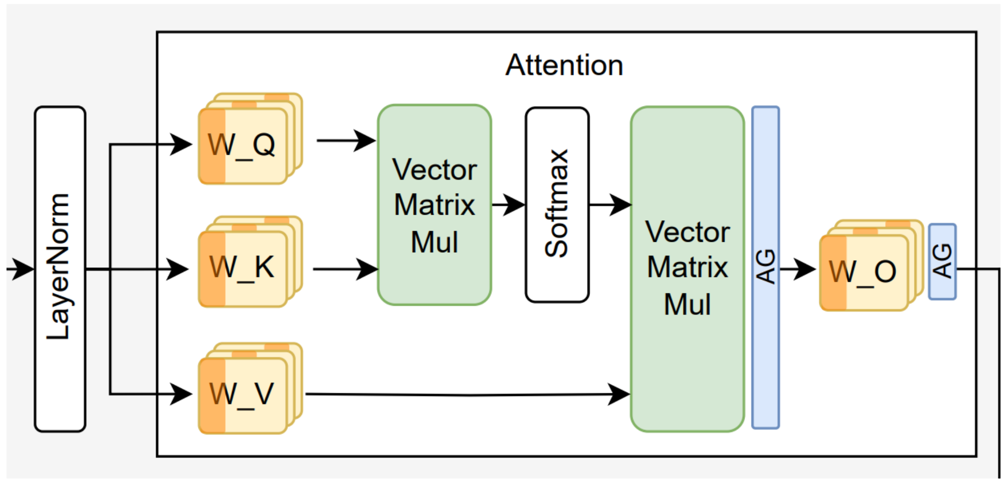
- Llama-2 70B on 8 x A100 (TP8)
  - Compute ≈ 280 TFLOPS/GPU
  - P_{Model} = 70B
  - $\text{Throughput}_{\text{optimal}}$ = 2000 tokens/s/GPU
    - the paper wrote 1857 tokens/s/GPU
    - different through, the conclusion does not change
  - current systems are not close to optimal throughput
    - resource has been wasted



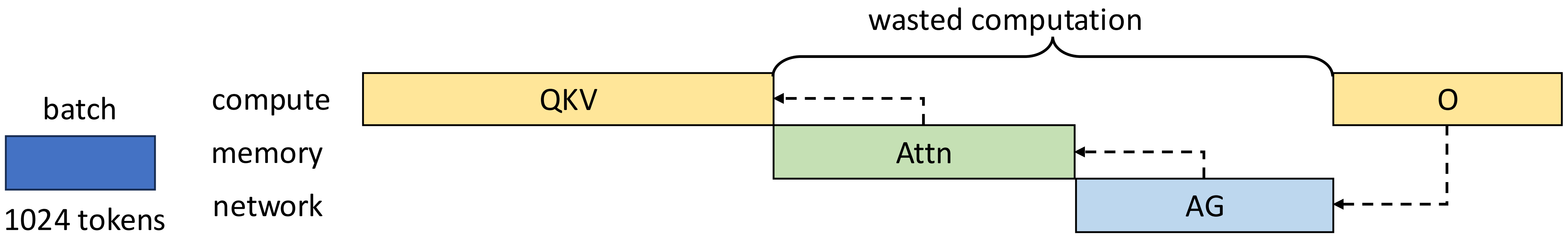
Current systems are far from optimal

Problem with Existing Solutions

- Sequentially execute different ops
  - ◆ waste computation resource



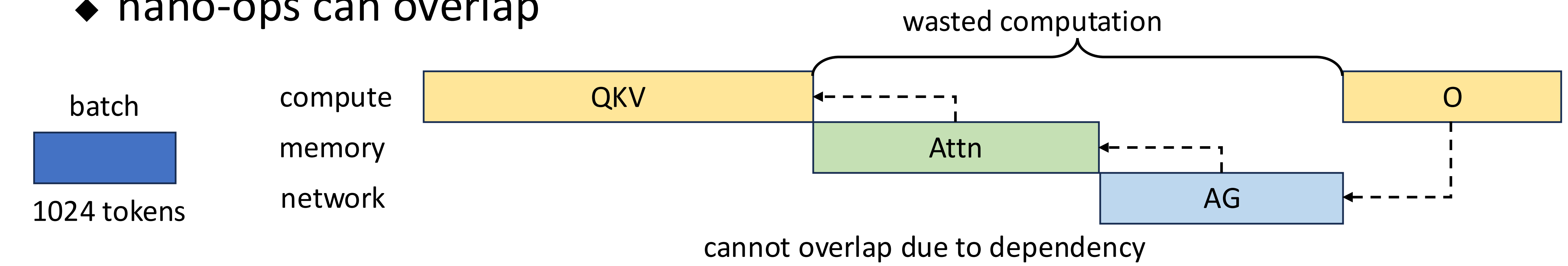
ops in attention block

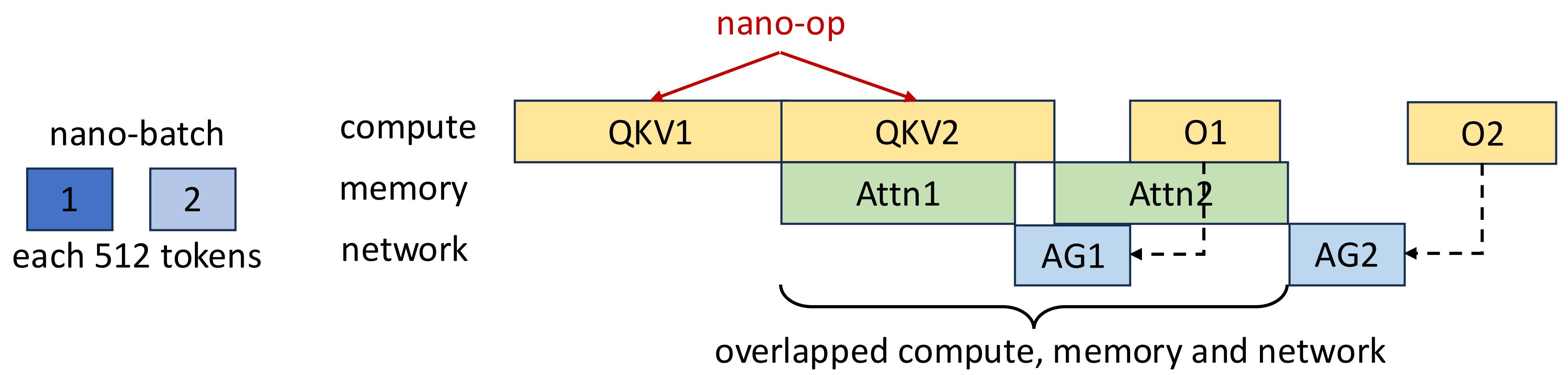


Current systems cannot overlap due to dependency

Intra-device Parallelism

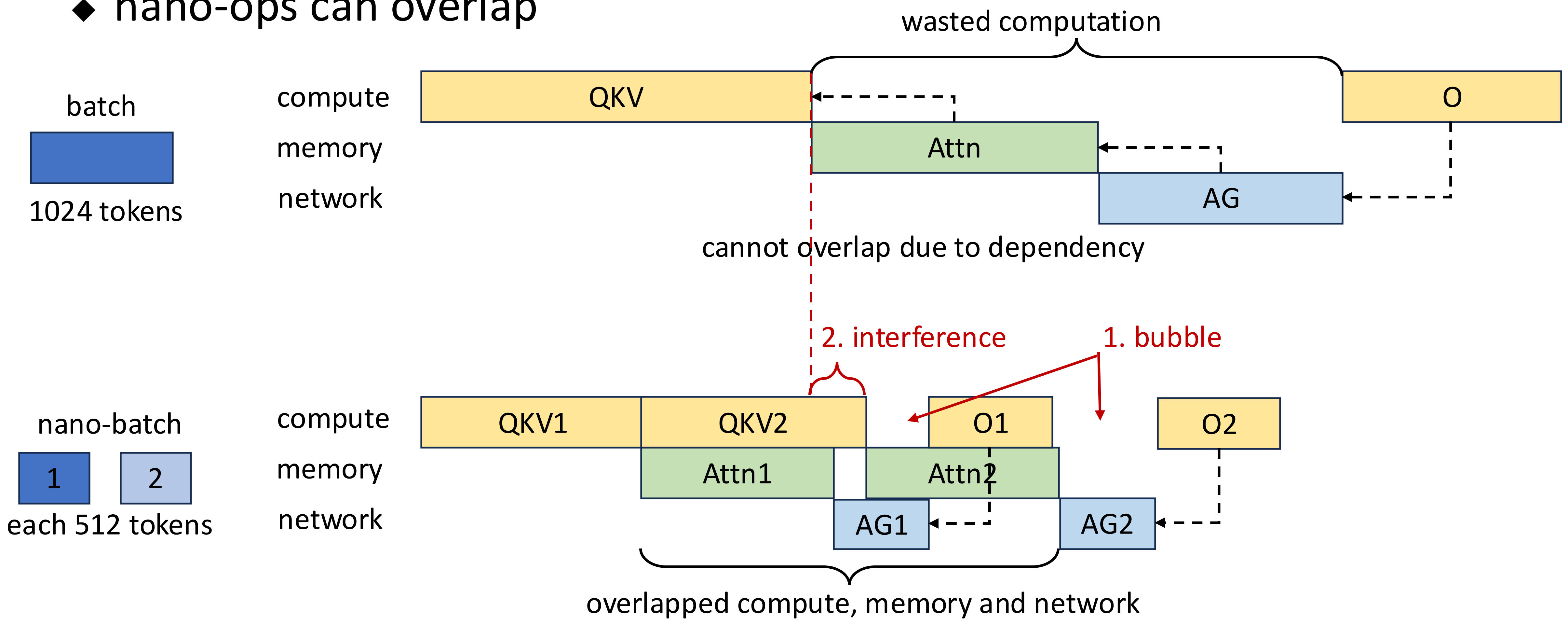
- Split batch -> nano-batch, op -> nano-op
  - nano-ops can overlap





Intra-device Parallelism

- Split batch -> nano-batch, op -> nano-op
  - nano-ops can overlap

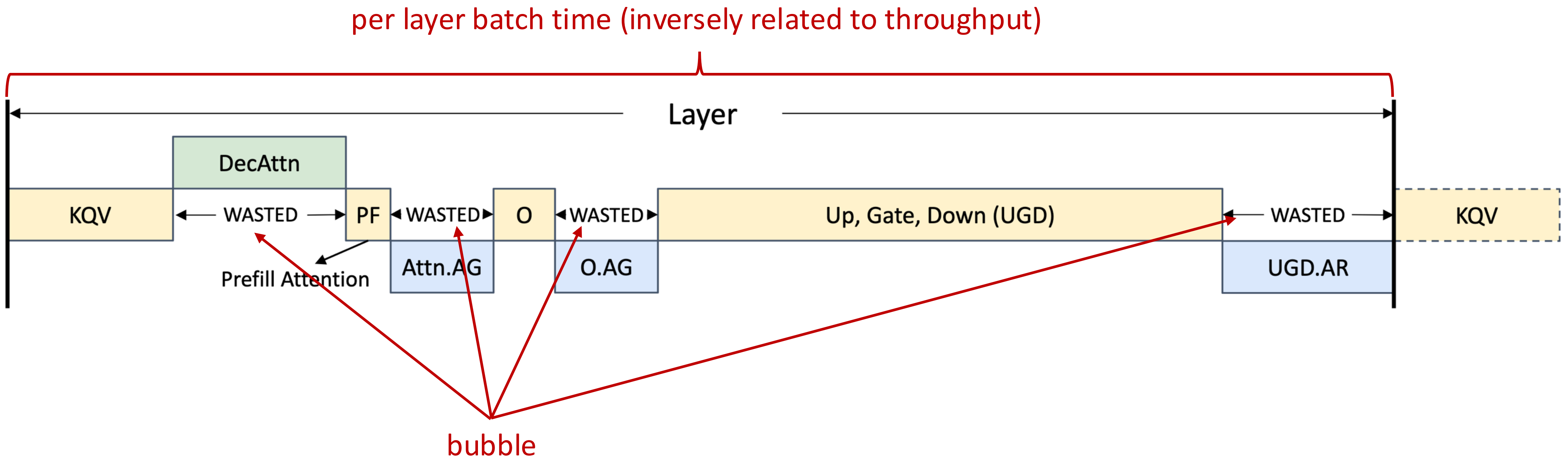


Outline

- Background
- Analysis
- Design & Implementation
  - Goal and challenges
  - Problem decomposition and solution
  - Nanoflow runtime
- Evaluation

Design

- Goal: automatic nano-batch pipeline computation
  - reduce bubble and improve throughput

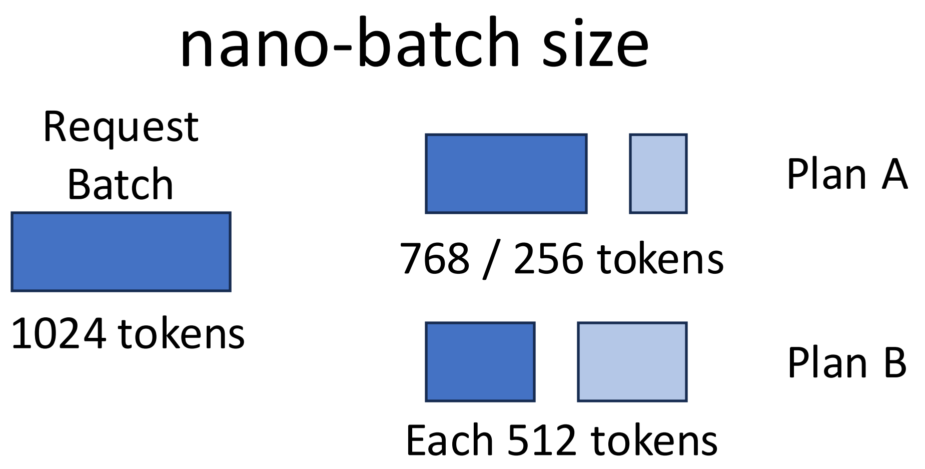


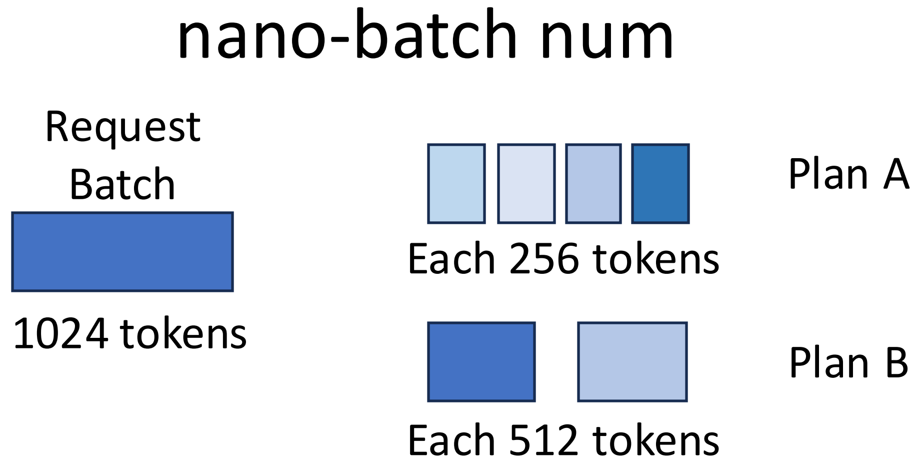
Automated Pipeline Search

- Challenges
  - large search space
    - nano-batch size & nano-batch num
    - nano-operation ordering

Automated Pipeline Search

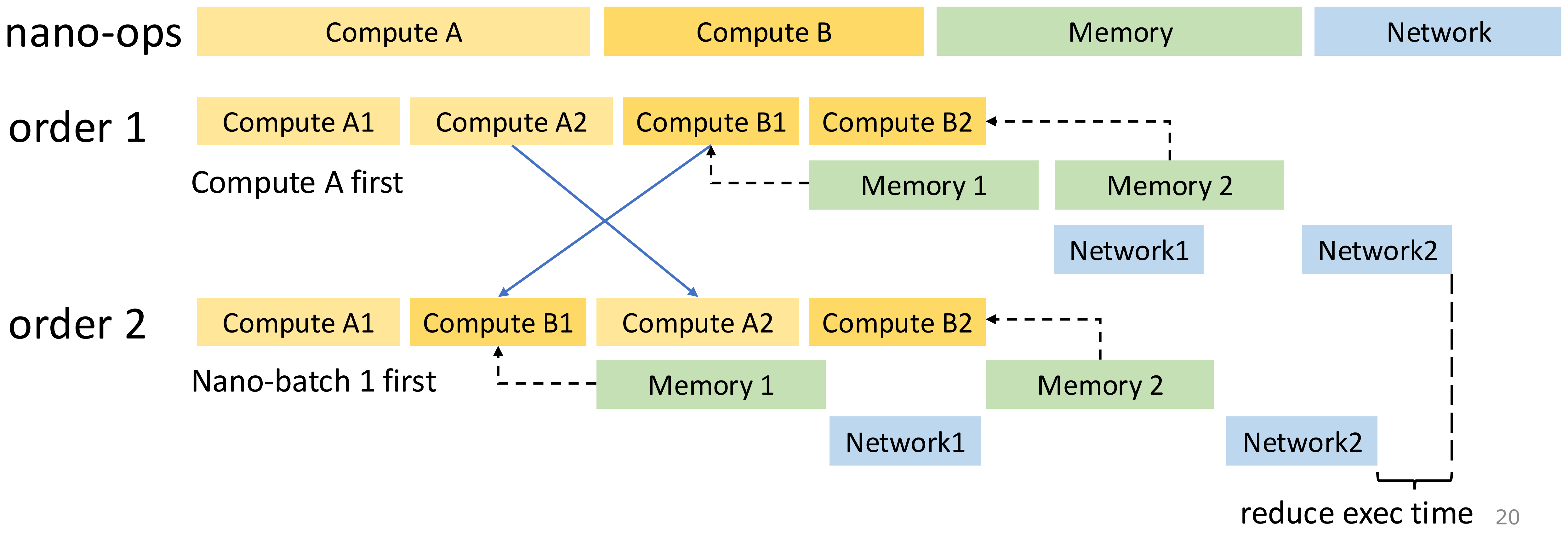
- Search Space
  - nano-batch size and number





Automated Pipeline Search

- Search Space
  - nano-batch size and number
  - nano-operation ordering



Automated Pipeline Search

- Challenges
  - large search space
    - nano-batch size & nano-batch num
    - nano-operation ordering
  - unpredictable performance of overlapped OPs
    - overlapped OPs are slower due to interference
    - overlapped performance is highly related to resource allocation
      - how to allocate resource

Automated Pipeline Search

- A 2-stage solution
- Preparation
  - profile kernel performance with/without interference
- Stage 1
  - ignore interference and build an ideal pipeline
  - minimizing bubbles as much as possible
- Stage 2
  - consider interference
  - adjust resource allocation in the pipeline
  - minimizing the overall execution time

Preparation

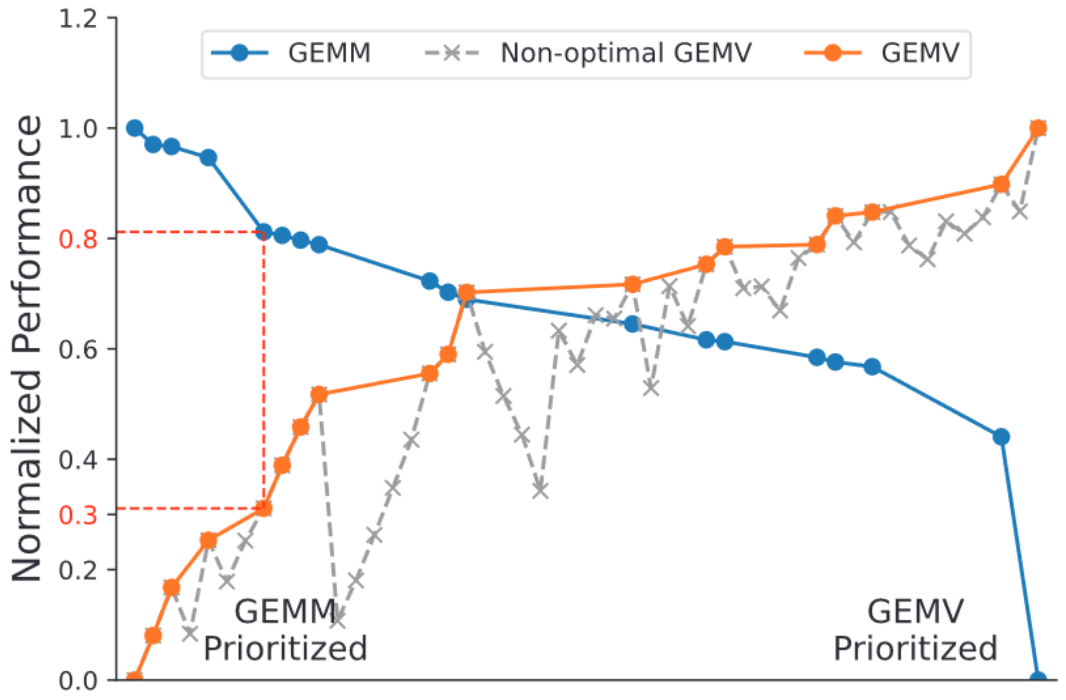
- Profiling **without interference**
  - goal: characterize the runtime of kernels **without interference**
  - find **max dense batch size**
    - fits model weights + KV-cache
  - profile kernels from **batch 128 → max batch** (step 128)
    - hardware friendly
    - reduce search space
  - output: table (operation Op, batch size B) → **runtime T**

Preparation

- Modeling overlapped kernel performance
  - definitions
    - P: normalized performance (overlapped performance / peak performance)
    - $R_{physical}$: physical resource fraction (cannot directly control due to NV’s API)
    - R: normalized GEMM performance as a proxy for $R_{physical}$
  - launch GEMM + GEMV (memory-bound) kernel pairs concurrently
    - measure
      - GEMM normalized performance (P_{GEMM})
      - GEMV normalized performance (P_{GEMV})
    - calculate : $R_{GEMM} = P_{GEMM}$; $R_{GEMV} = 1 – R_{GEMM}$
    - vary number of thread blocks for GEMM
    - R->P mapping for GEMM and GEMV
  - R->P mapping for network is profiled in the same way

Preparation

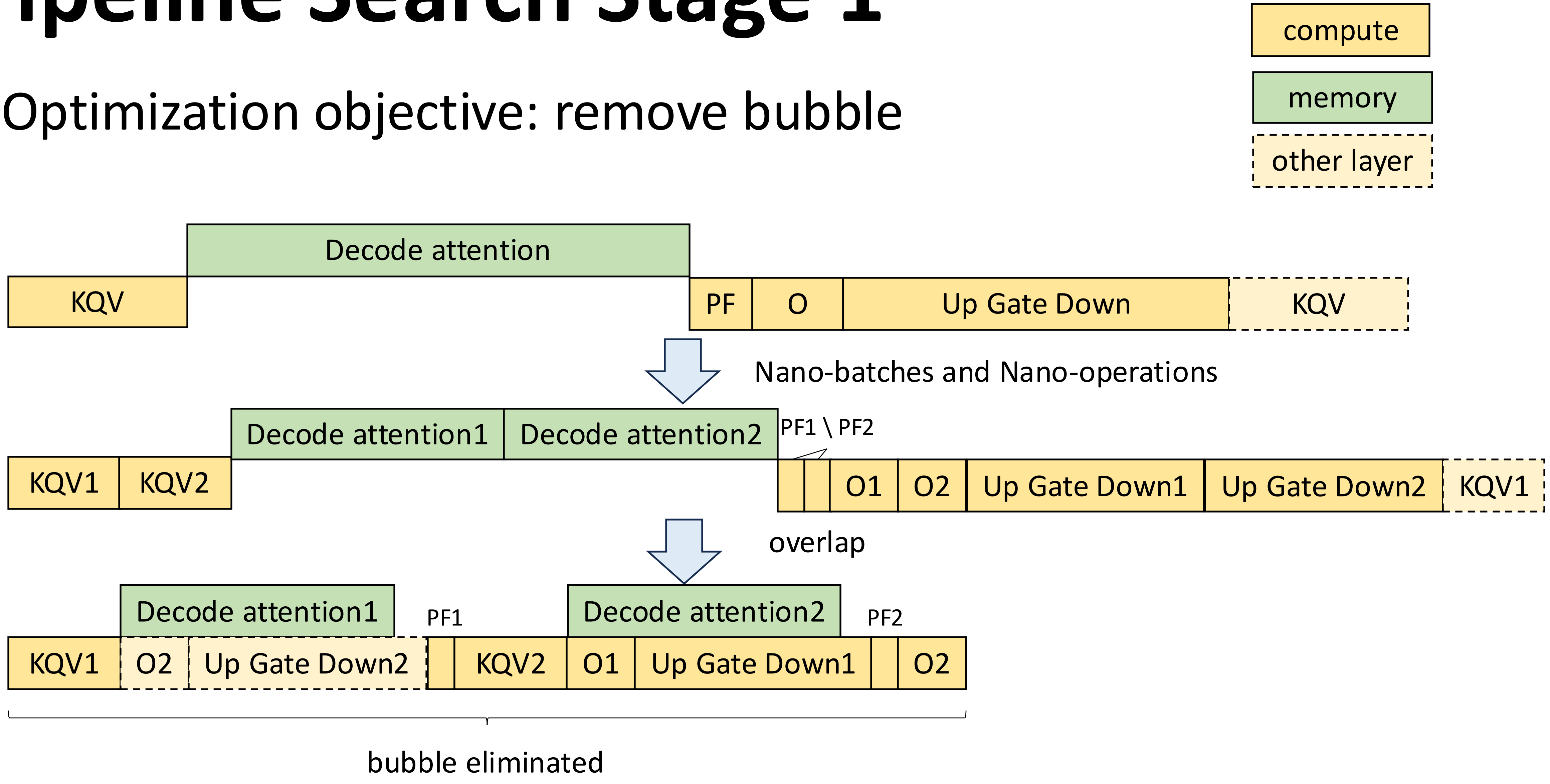
- Modeling
  - we obtain the mapping from R to P
  - $T_{overlap} = T_{single} / P(R)$, we can get performance from resource allocation



grey points are under performance

Pipeline Search Stage 1

- Optimization objective: remove bubble

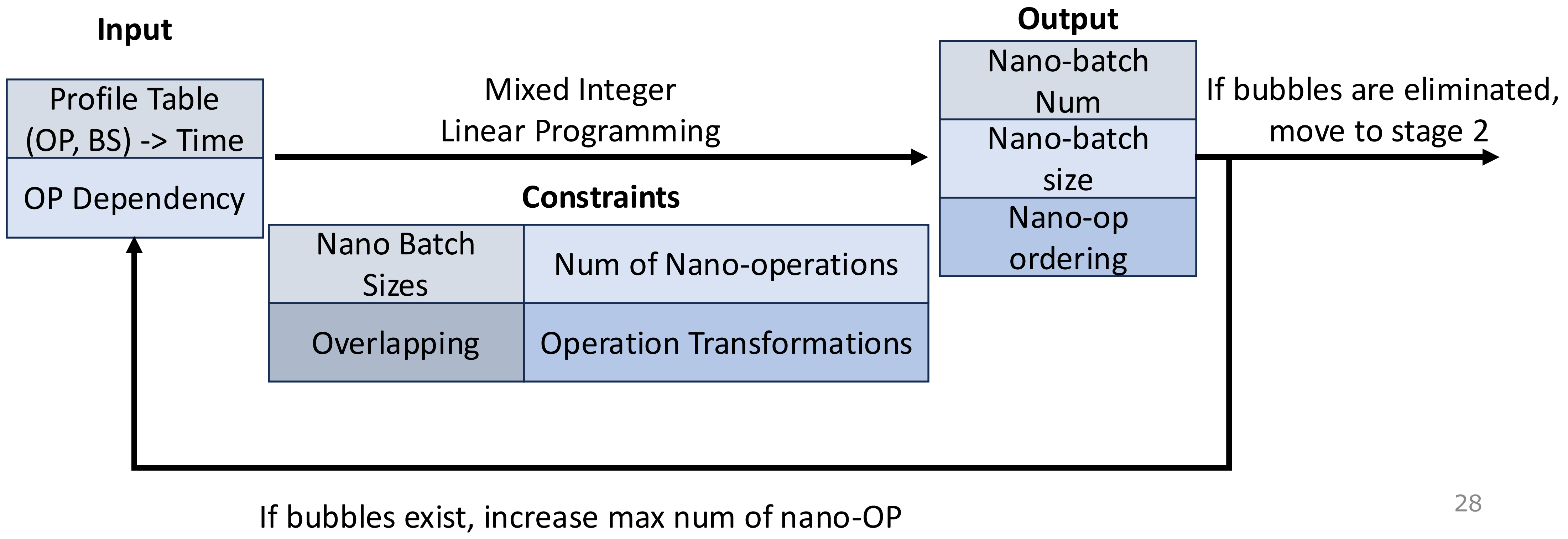


Pipeline Search Stage 1

- Constraints
  - batch size: from 128 to dense batch size(step 128)
  - dependency: compute dependency defined in the PyTorch code
  - overlapping: overlapping kernels constrained by the same resource are useless
  - operation transformations: collectives can be rewritten into equivalent forms
  - num of nano-OPs: cannot exceed the given maximum
    - to restrict overhead

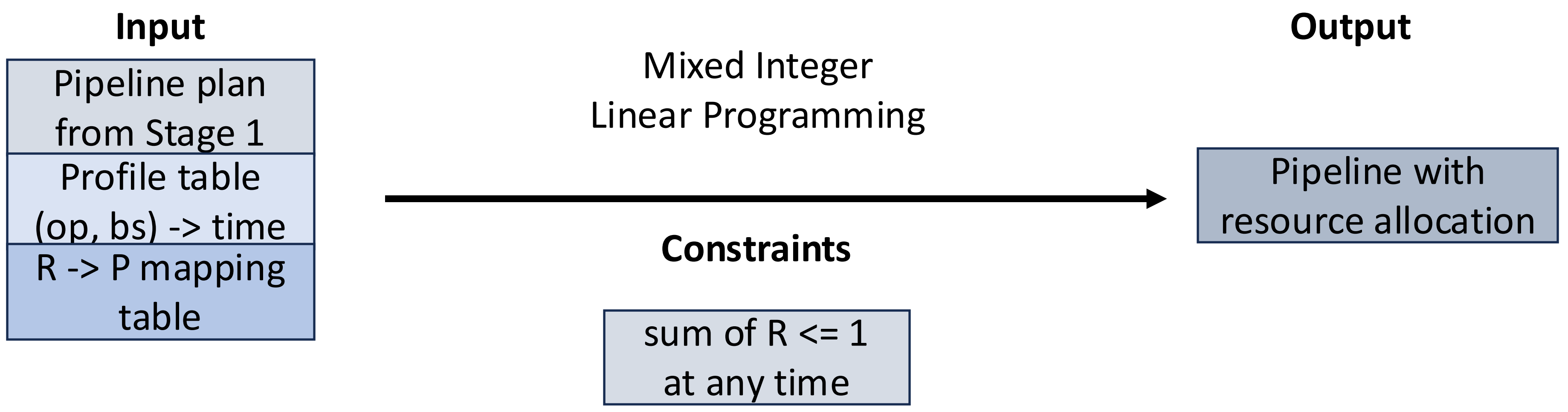
Pipeline Search Stage 1

- Method: MILP
  - MILP: choose continuous and integer variables under linear constraints to optimize a linear objective
  - Goal: minimize bubble


Input
Profile Table (OP, BS) -> Time
OP Dependency
Mixed Integer Linear Programming
Constraints
Nano Batch Sizes
Num of Nano-operations
Overlapping
Operation Transformations
Output
Nano-batch Num
Nano-batch size
Nano-op ordering
If bubbles are eliminated, move to stage 2
If bubbles exist, increase max num of nano-OP

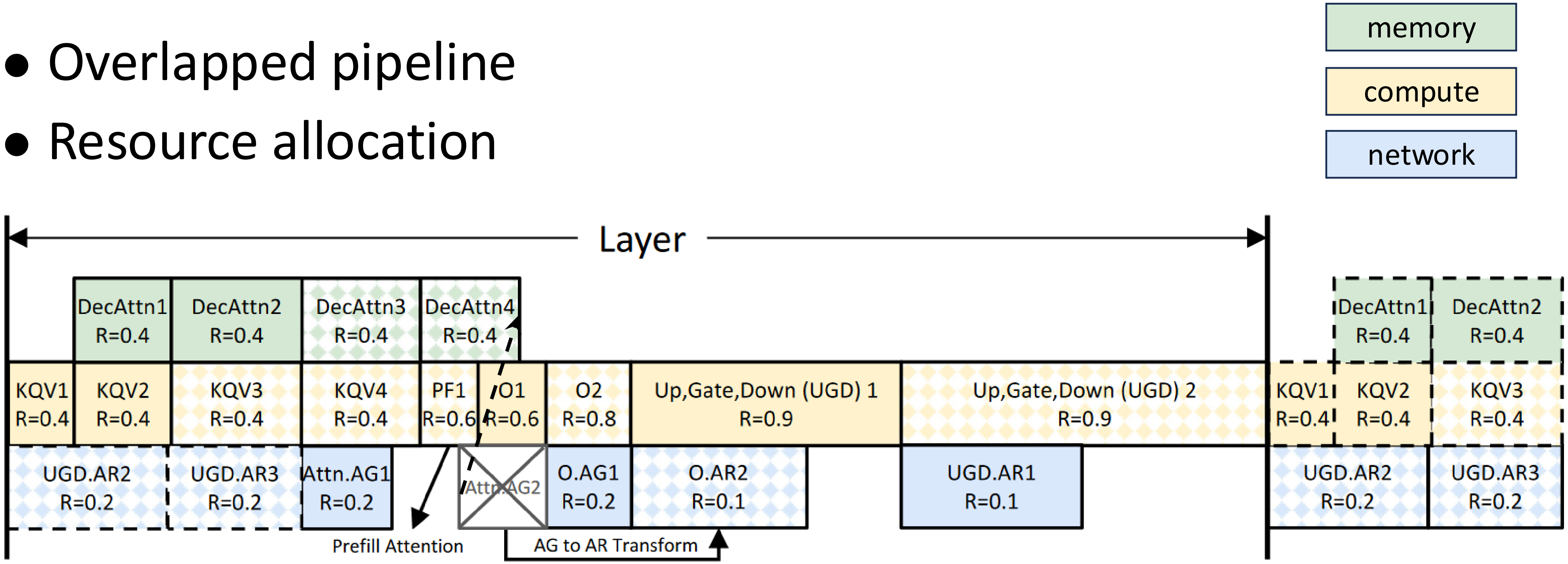
Pipeline Search Stage 2

- Goal: minimize pipeline execution time
  - refines pipeline by considering kernel interference


Input
Pipeline plan from Stage 1
Profile table (op, bs) -> time
R -> P mapping table
Mixed Integer Linear Programming
Constraints
sum of R <= 1 at any time
Output
Pipeline with resource allocation

Automated Pipeline Search

- Overlapped pipeline
- Resource allocation



The solid background and shaded background represents input batch 0-768 and 768-2048

Nanoflow Runtime

- Asynchronous requests scheduling:
  - overlap CPU scheduling with GPU computing
    - to hide scheduling overhead
    - method: defer request completion check by one iteration
      - at the cost of at most one extra decode token
      - acceptable compared to hundreds of decode tokens
- KV-cache management
  - offloads KV-cache to CPU DRAM + SSD
  - to support multi-round conversations

Outline

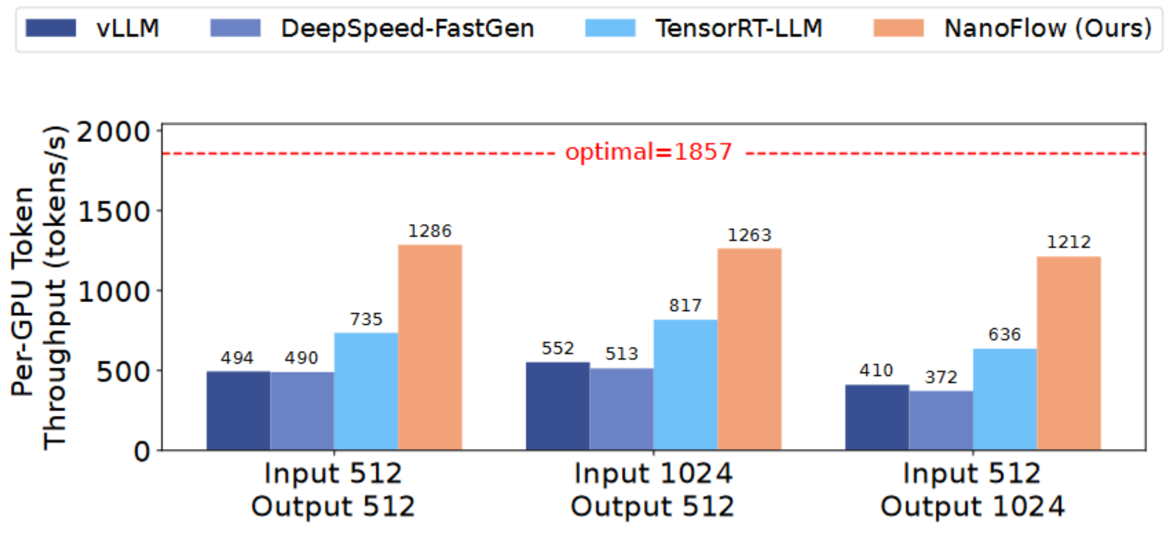
- Background
- Analysis
- Design & Implementation
- Evaluation

Evaluation

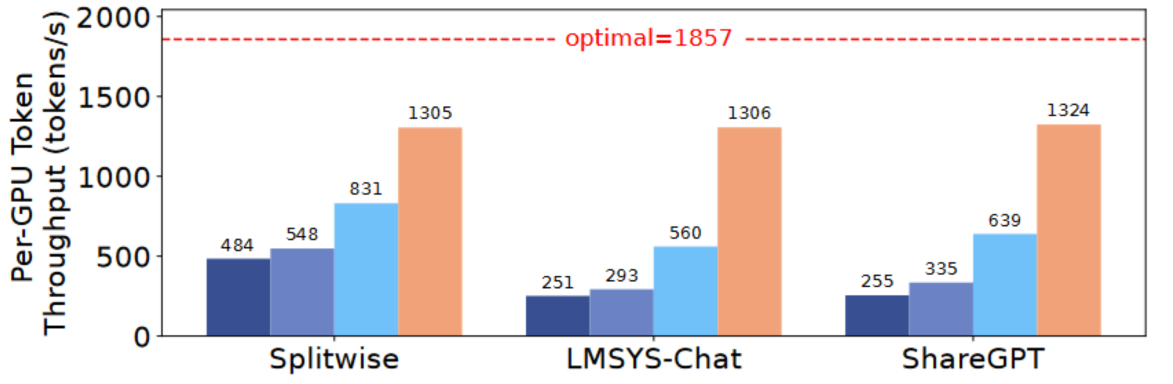
- Setup
  - hardware: 8x A100 80GB SXM GPUs interconnected via NVLink 600GB/s
  - models: LLaMA-2-70B
  - datasets: Splitwise, LMSYS-Chat-1M, ShareGPT
- Baselines
  - DeepSpeed-FastGen
  - TensorRT-LLM
  - vLLM
  - all baselines enables chunked prefill

Throughput

- Offline throughput comparison
  - constant-length workloads
    - at least 1.73x throughput gain
  - real traces
    - at least 1.91x throughput gain
  - intra-device overlap with nano-ops
    - split ops into nano-ops and overlap compute-, memory-, and network-bound kernels on the same GPU
    - pipeline search chooses useful overlap and remove compute bubbles



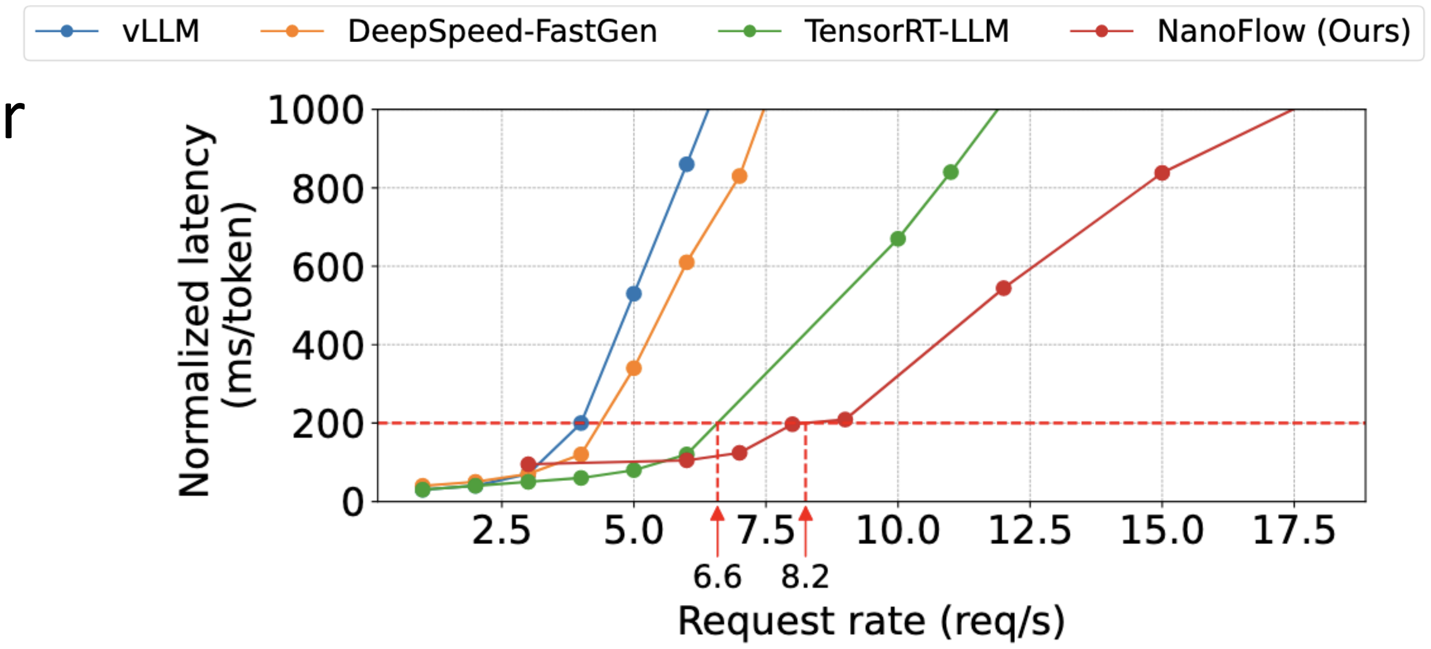
(a) LLaMA-2-70B, 8 GPU, TP=8, Constant Input & Output Length



(b) LLaMA-2-70B, 8 GPU, TP=8, Input & Output Length from Dataset

Latency

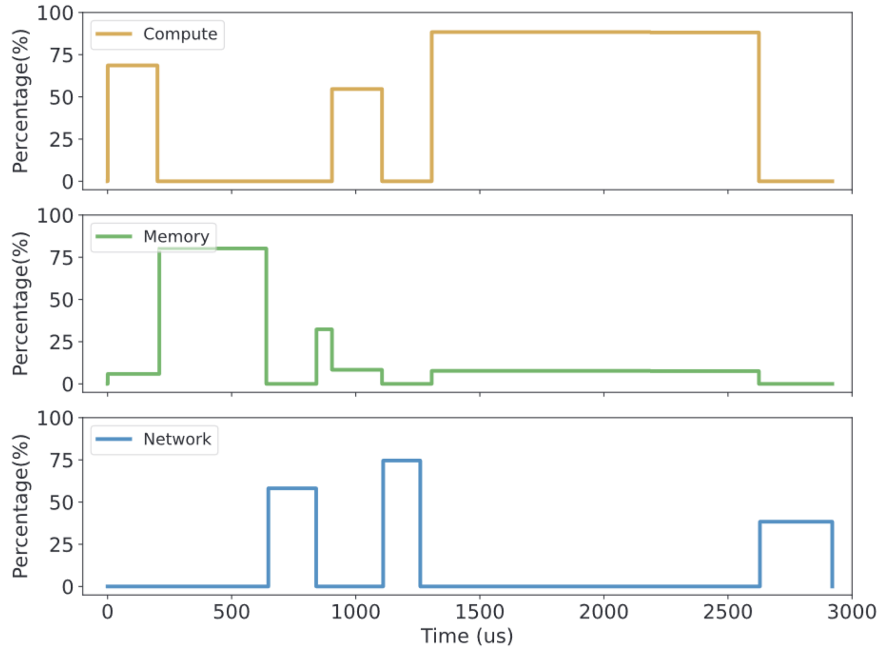
- Online latency
  - sustain low latency under 1.24x higher QPS
  - other datasets show similar improvements
  - latency under low QPS is higher
    - because NanoFlow uses large batch size



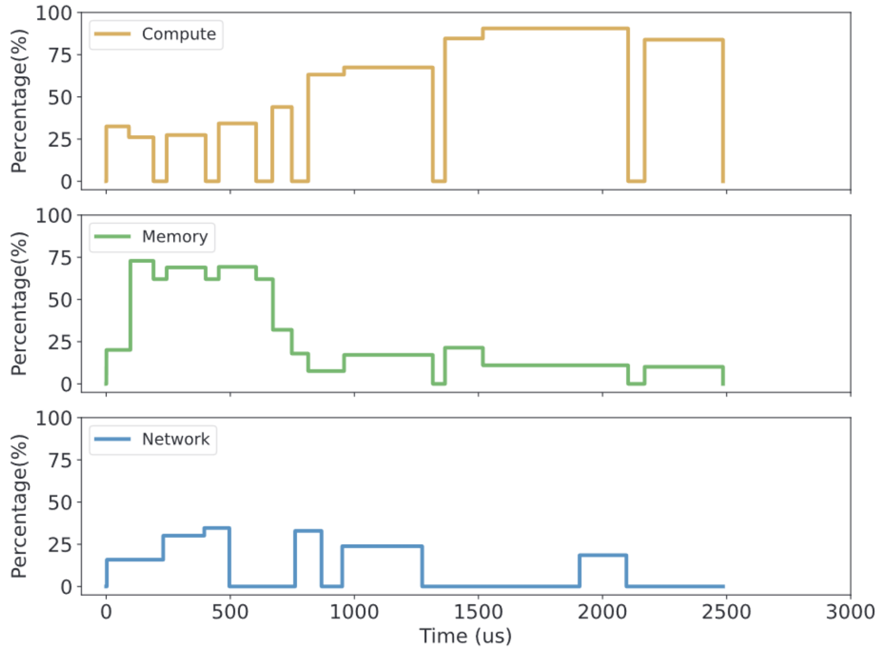
Normalized latency on Splitwise dataset

Evaluation

- Resource usage
  - ◆ Nanoflow utilizes multiple resources concurrently, achieving higher usage



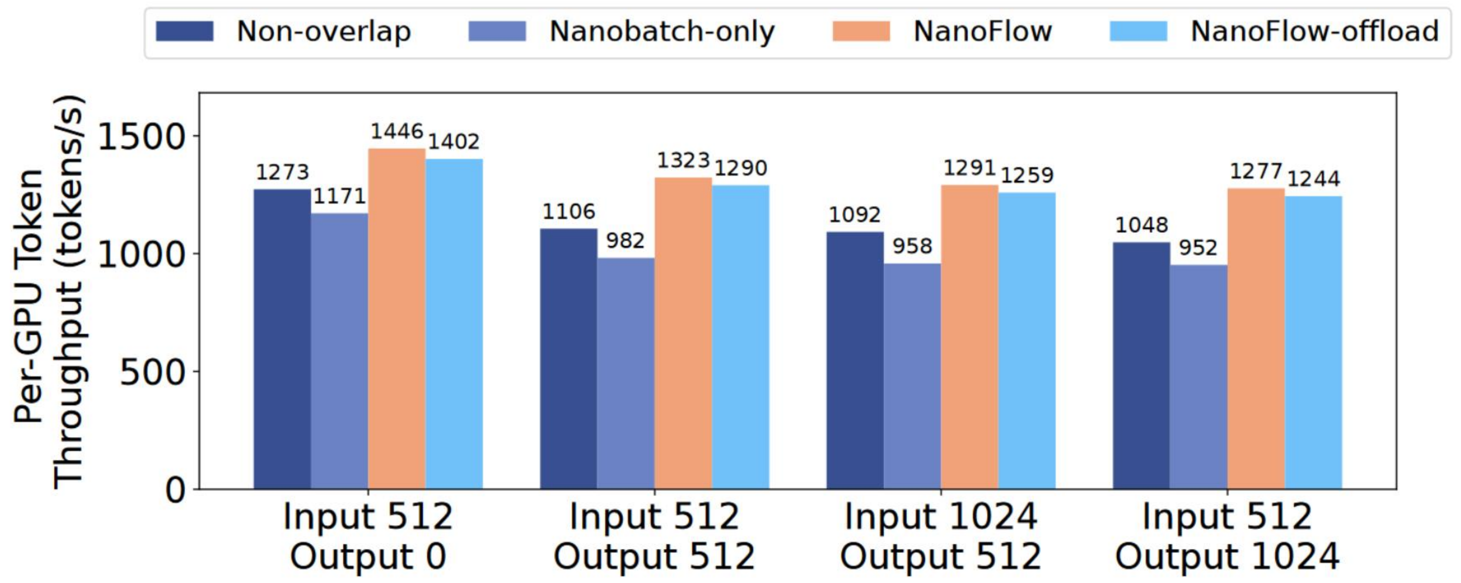
(a) Non-overlap pipeline resource usage



(b) NanoFlow resource usage

Evaluation

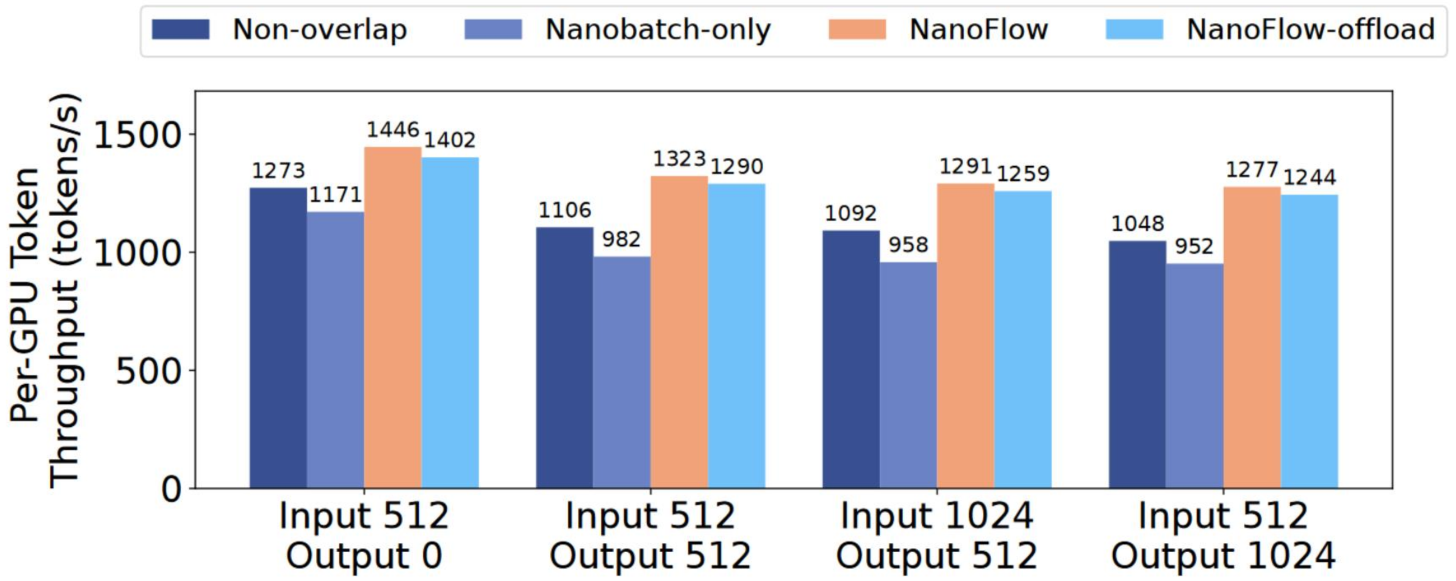
- Ablation study
  - 13.2% overhead from splitting into nano-batches alone
  - 1.07× speedup by overlapping network-bound kernels
  - 1.17× speedup by overlapping both network- and memory-bound



network + compute network + memory + compute

Evaluation

- Ablation study
  - KV offloading only causes a small throughput loss (3%)
    - reduces compute by 3.02x on multi-round LMSYS-chat (not shown in figure)



Conclusion

- Highlights
  - theoretically analyzed LLM inference characteristics and bottlenecks
  - proposed overlapping use of different resources
  - designed a scheme for allocating heterogeneous resources in parallel kernels
- Remaining problems
  - very large batches cannot meet low-latency needs (e.g., 100 ms)
  - compute-bound assumption fails for long decoding