NanoFlow: Towards Optimal Large Language Model Serving Throughput

Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Qinyu Xu, Tian Tang, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Stephanie Wang, Arvind Krishnamurthy, Baris Kasikci

LLMs Enable Many New Applications

AI Assistants (ChatGPT, Google Bard) 

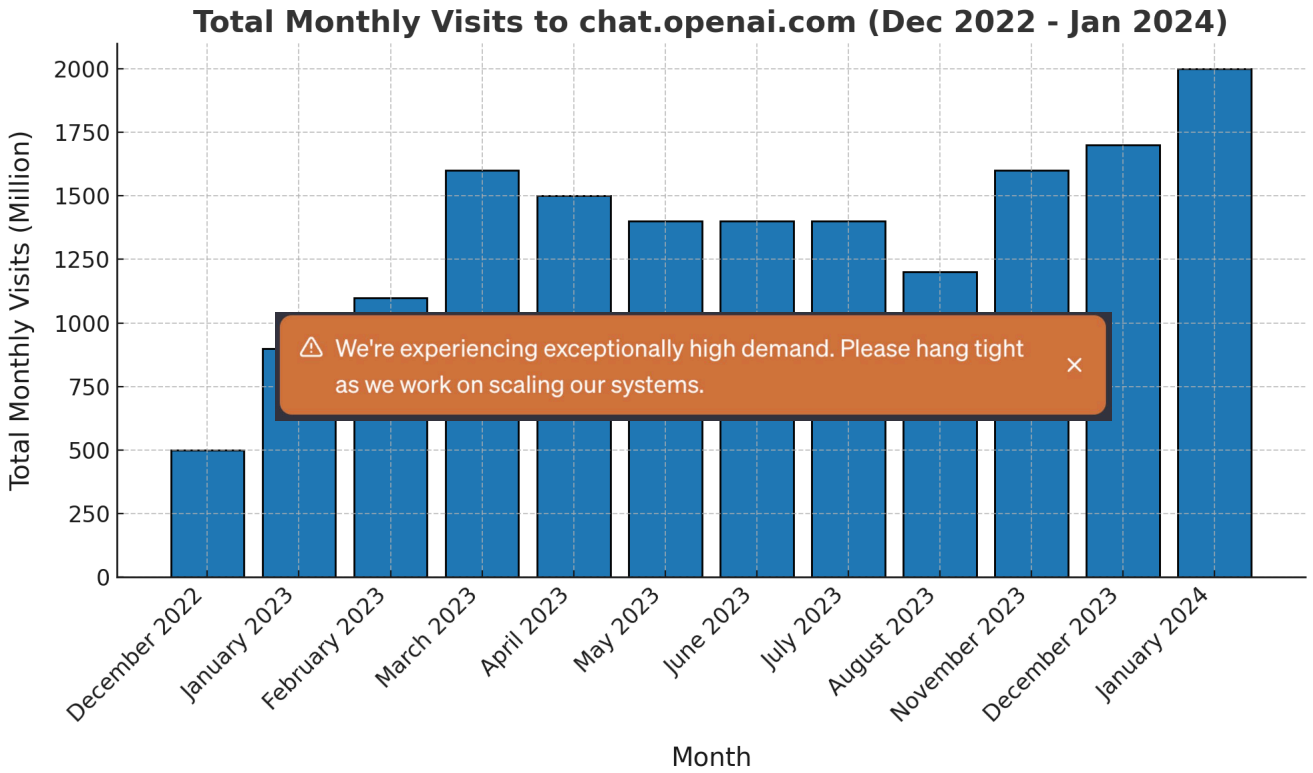
Text-to-Image (DALL·E, MidJourney) 

Creative Writing (Jasper, Copy.ai) 

AI Coding Tools (GitHub Copilot, Replit) 

Text-to-Speech & Audio (Descript, Synthesia) 

LLM Serving Systems Face Surging Demand

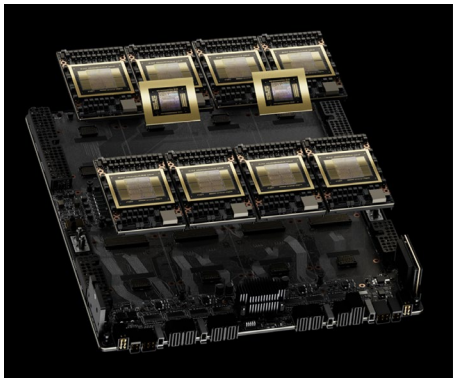


Source: https://nerdynav.com/chatgpt-statistics/

Cost of Training and Serving LLMs is Massive

Large-scale investment in H100s in 2024

- Meta: 300K[1]
- Google: 150K[1]
- Microsoft: 150K[1]
- X: 85K[2]



NVIDIA H100 HGX Server

- ~ $300,000
- High operating cost
  - 8000 W
- Up to ~11 months lead time

[1] Tom's Hardware
[2] The Register

Batching user requests and aiming for optimal throughput are key

NanoFlow

What is the optimal LLM serving throughput?

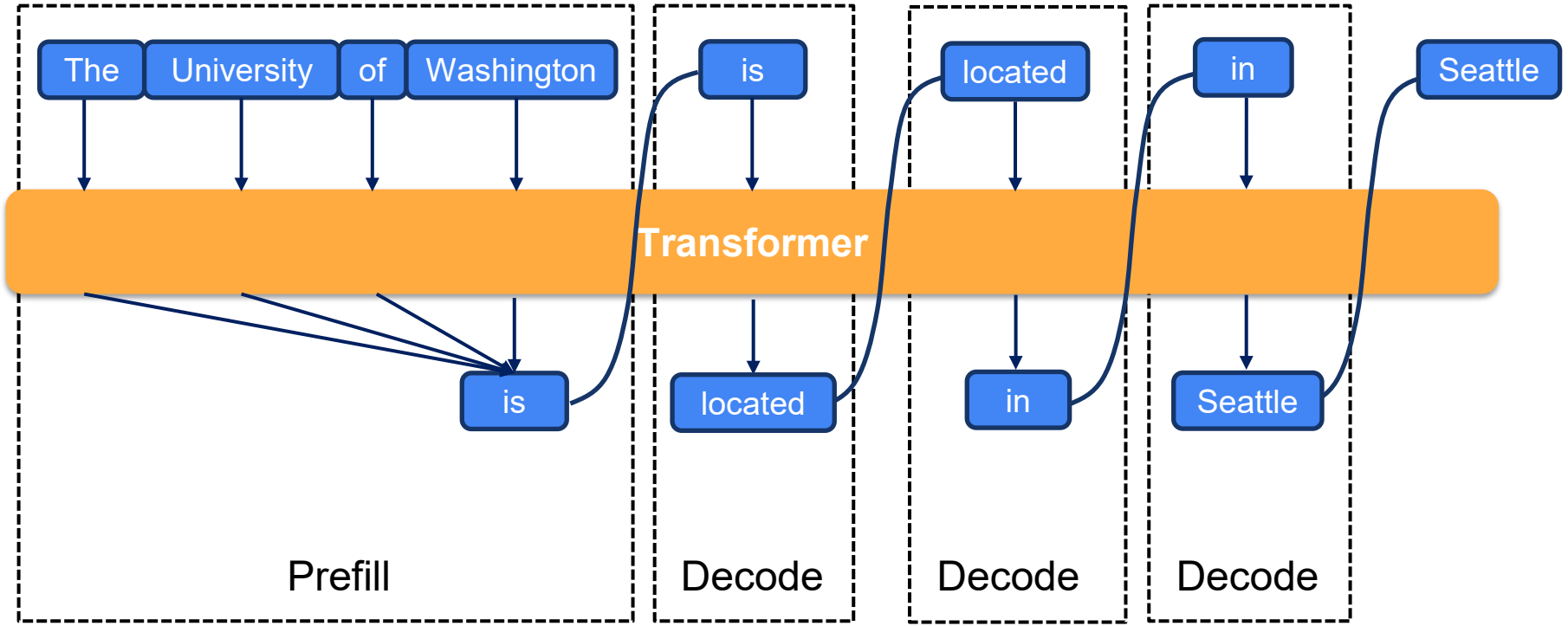
- Operation Classification
- Resource Bottleneck analysis

How to approach optimal LLM serving?

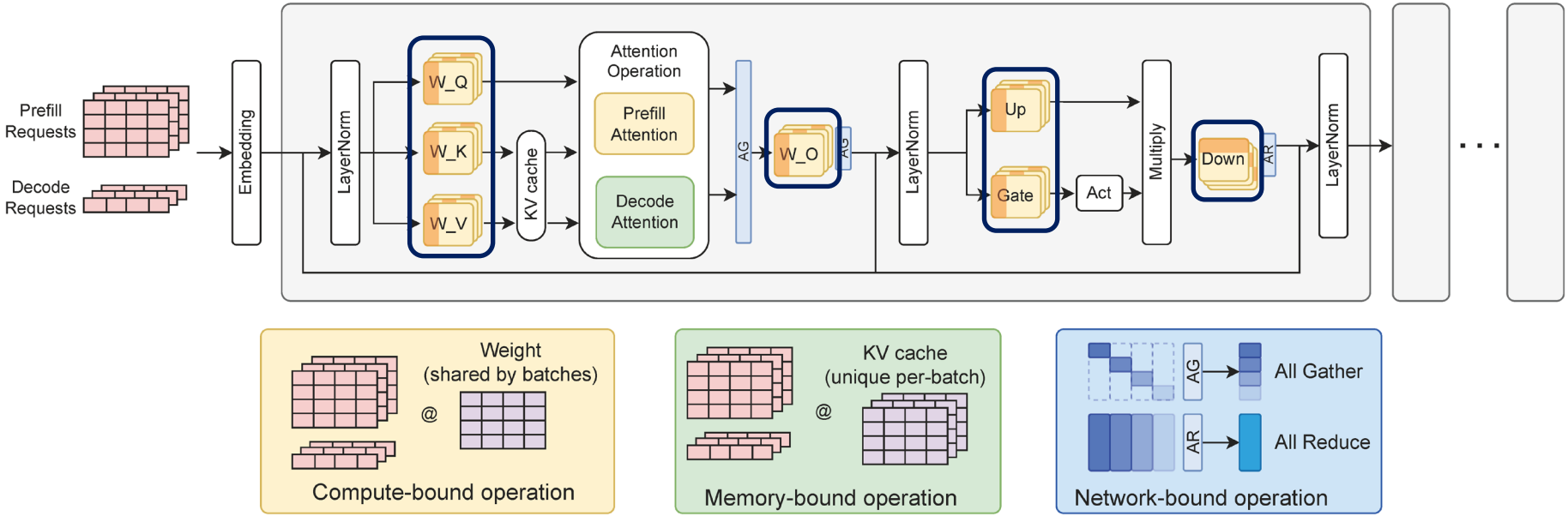
- Nanobatch
- AutoSearch
- Async Scheduling
- KV cache management

How does Nanoflow compare with existing frameworks?

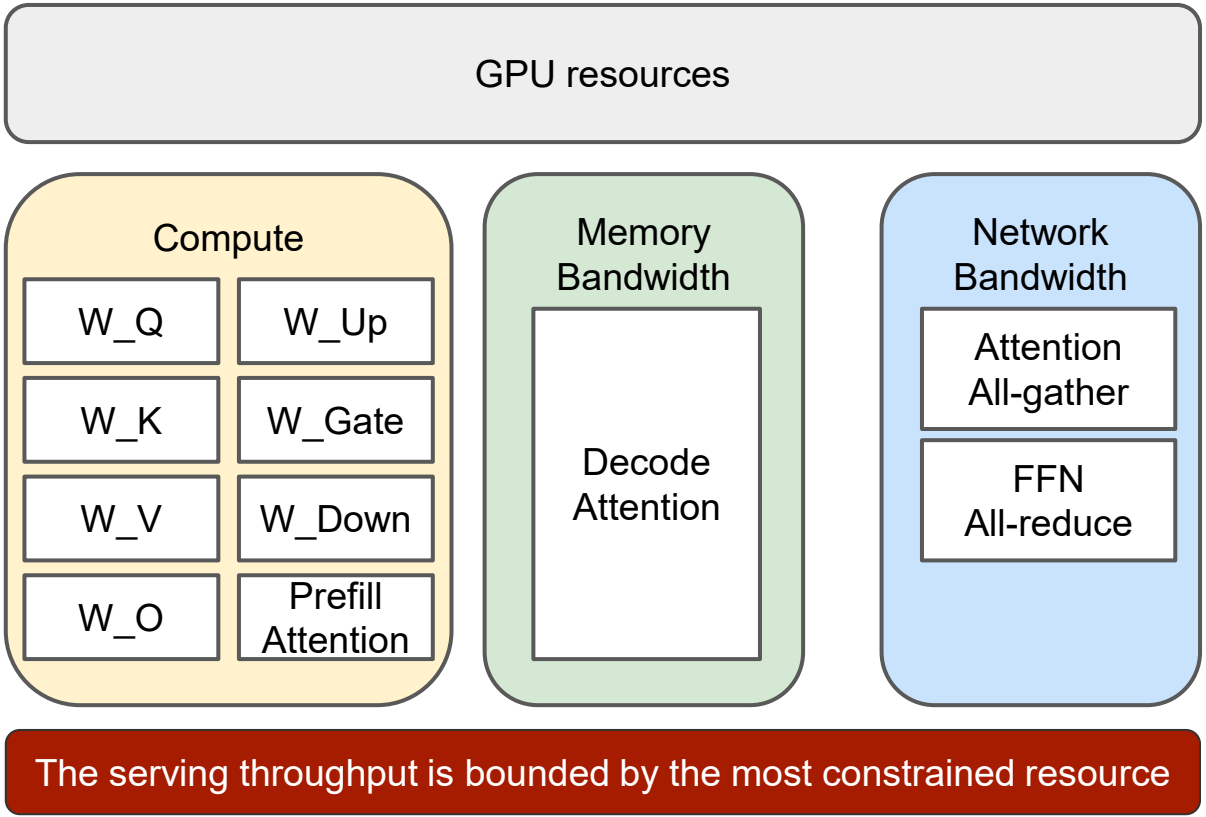
Prefill and Decode Phases of Transformer



LLM inference have variety of operations



Matching the resource needs of operations with GPU resources



Memory Time Estimation

Whole GPU memory would be loaded once during one iteration

$$T_{memory} = \frac{GPUmem}{MemBw}$$

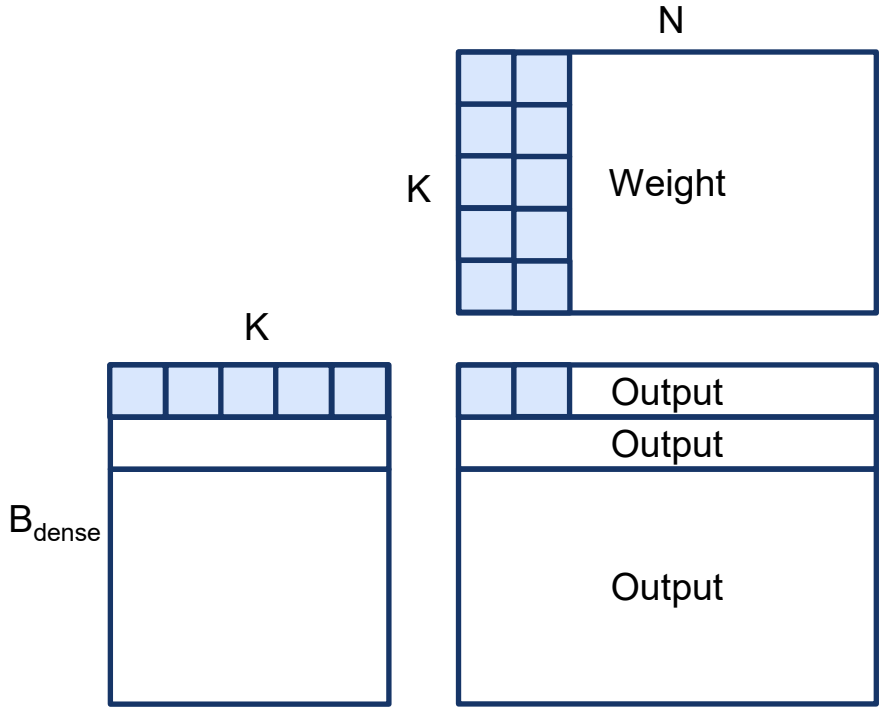
Compute Time Estimation

Every GEMM compute = $2\ B_{dense}\ K\ N$

All Dense Operations

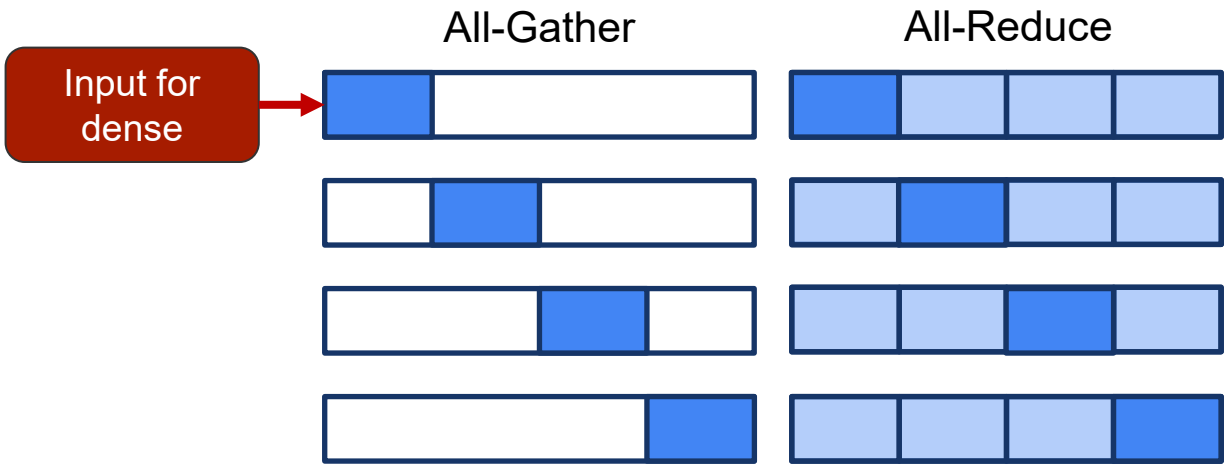
$$2\ B_{dense}\ K_1\ N_1 + 2\ B_{dense}\ K_2\ N_2 + 2\ B_{dense}\ K_2\ N_2 + \dots\ \dots\ \dots\ \dots + 2\ B_{dense}\ K_i\ N_i$$

$$2\ B_{dense}\ P_{model}$$



$$T_{compute} = \frac{2B_{dense}P_{model}}{Compute}$$

Network Time Estimation



$$T_{net} = \frac{4(N-1)D_{model}B_{dense}LS_{type}}{NetBw}$$

Workload Classification

$$T_{memory} = \frac{GPUmem}{MemBw}$$

$$T_{compute} = \frac{2B_{dense}P_{model}}{Compute}$$

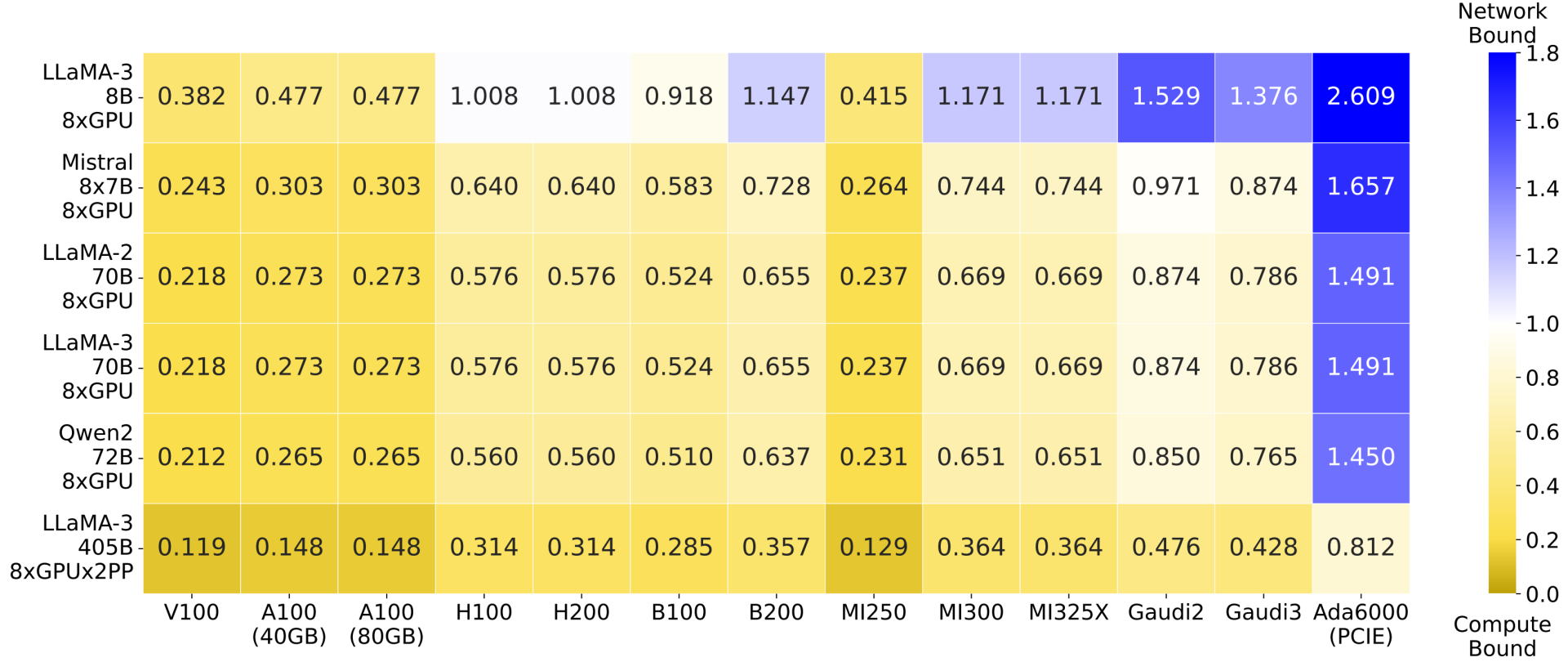
$$T_{net} = \frac{4(N-1)D_{model}B_{dense}LS_{type}}{NetBw}$$

Compare Net and Compute

$$\frac{T_{net}}{T_{compute}} = 2(N-1)\frac{D_{model}}{P_{model}}L\frac{S_{type}}{NetBw}Compute$$

Shift to Compute Bound	Shift to Network Bound	Non Relevant
P_{model}	$Compute$	B_{dense}
$NetBW$	S_{type}	

Compare Net and Compute



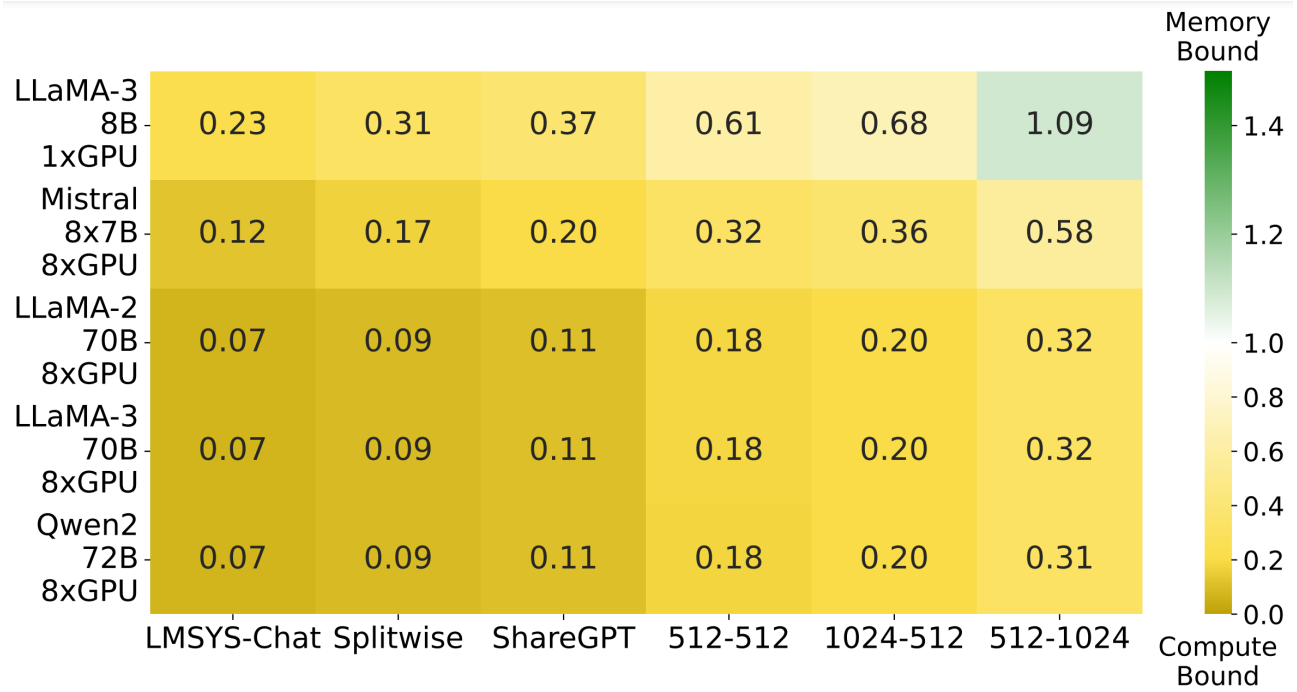
Compare Net and Compute

$$T_R = \frac{Compute}{MemBw} \frac{GPUmem}{2B_{dense}P_{model}}$$

$$T_R = \frac{Compute}{MemBw} \frac{GPUmem}{\frac{GPUmem}{S_{type}} - P_{model}} (d+1)\left(1 - \frac{d/2}{p+d}\right) \frac{D_{model}L}{R_{GQA}P_{model}}$$

Shift to Compute Bound	Shift to Memory Bound	Limited Effect
P_{model}	d	p
B_{dense}		

Compare Compute and Memory



High Utilization of GPU **compute** is the key for LLM serving

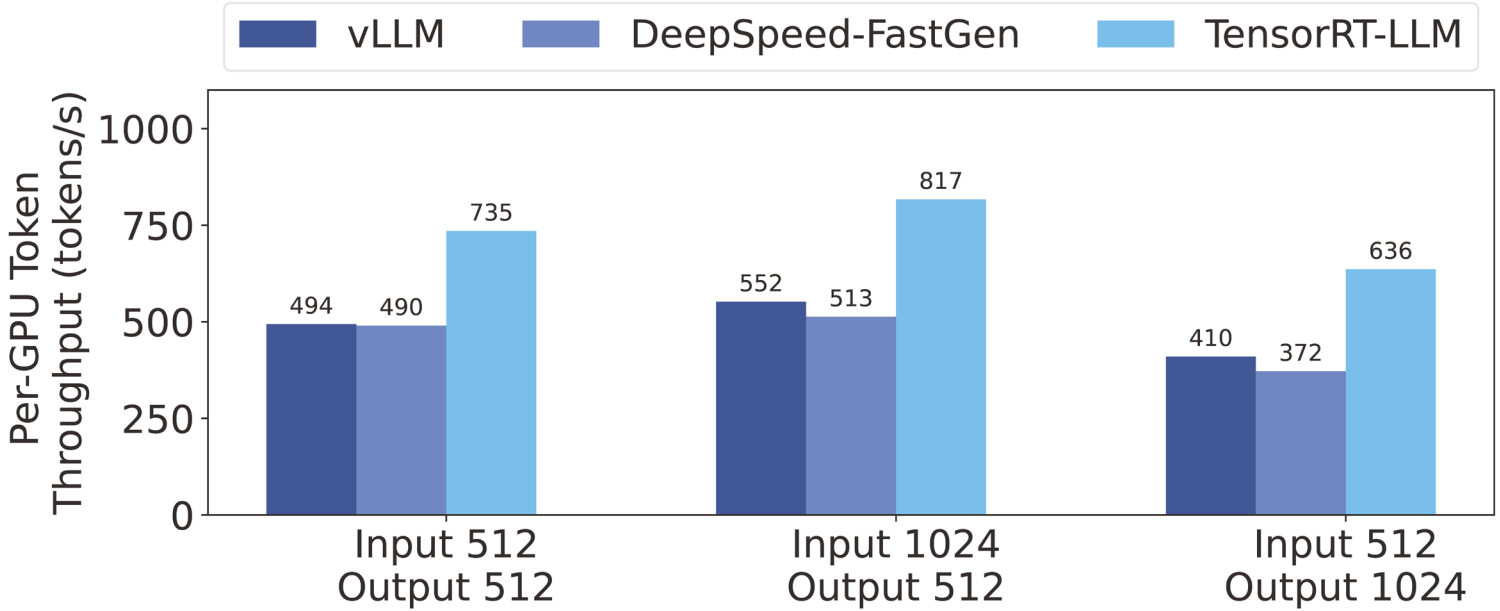
Optimal throughput

$$\text{Total Throughput} = \text{Decode} + \text{Prefill} = \frac{B_{decode} + B_{prefill}}{\text{Iteration Time}} = \frac{B_{dense}}{T_{compute}} = \frac{B_{dense}}{\frac{2B_{dense}P_{model}}{Compute}}$$

$$\text{Throughput} = \frac{Compute}{2P_{model}}$$

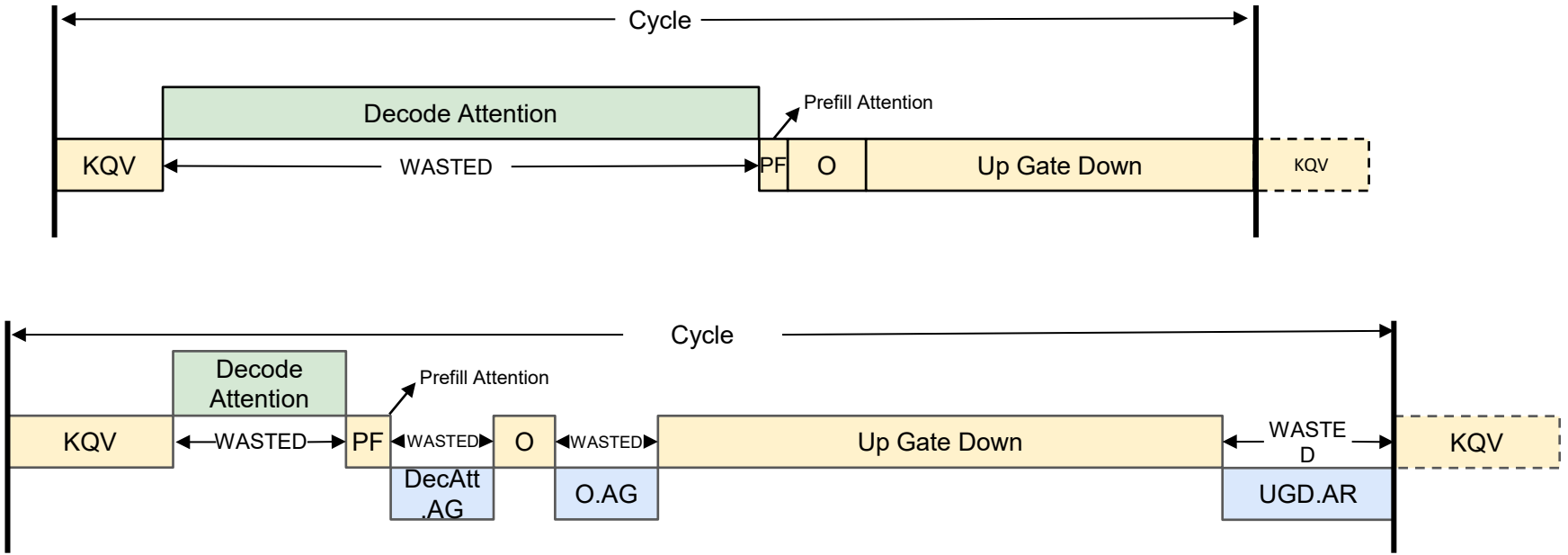
70B, A100 1857 token/s/GPU	8B, A100 16250 token/s/GPU	8*7B, Top2, A100 10304 token/s/GPU

Revisit the performance of existing frameworks



Large gap towards optimal throughput

Heterogenous resource demands cause inefficiency



Underutilize compute when perform network or memory bound operations

NanoFlow

What is the optimal LLM serving throughput?

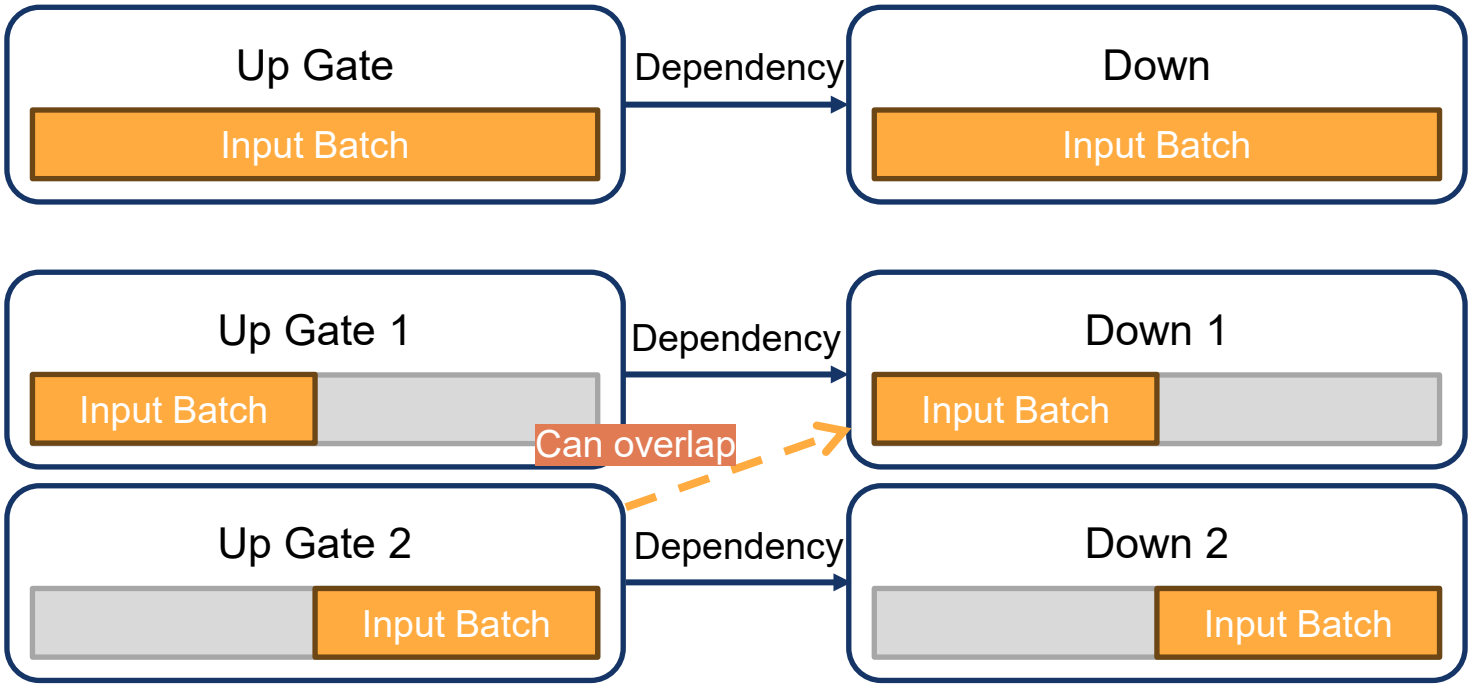
- Operation Classification
- Resource Bottleneck analysis

How to approach optimal LLM serving?

- Nanobatch
- AutoSearch
- Async Scheduling
- KV cache management

How does Nanoflow compare with existing frameworks?

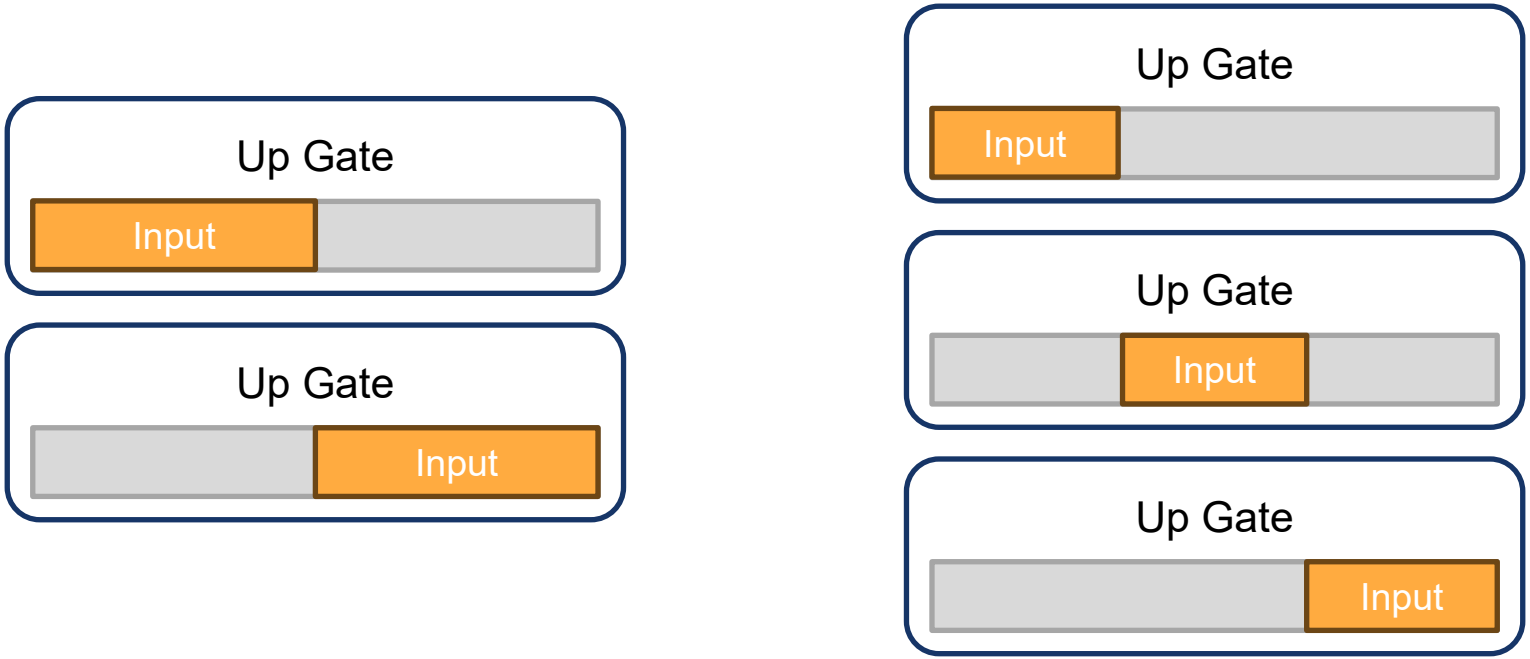
Nanobatching



Pay extra weight loading for overlapping opportunities

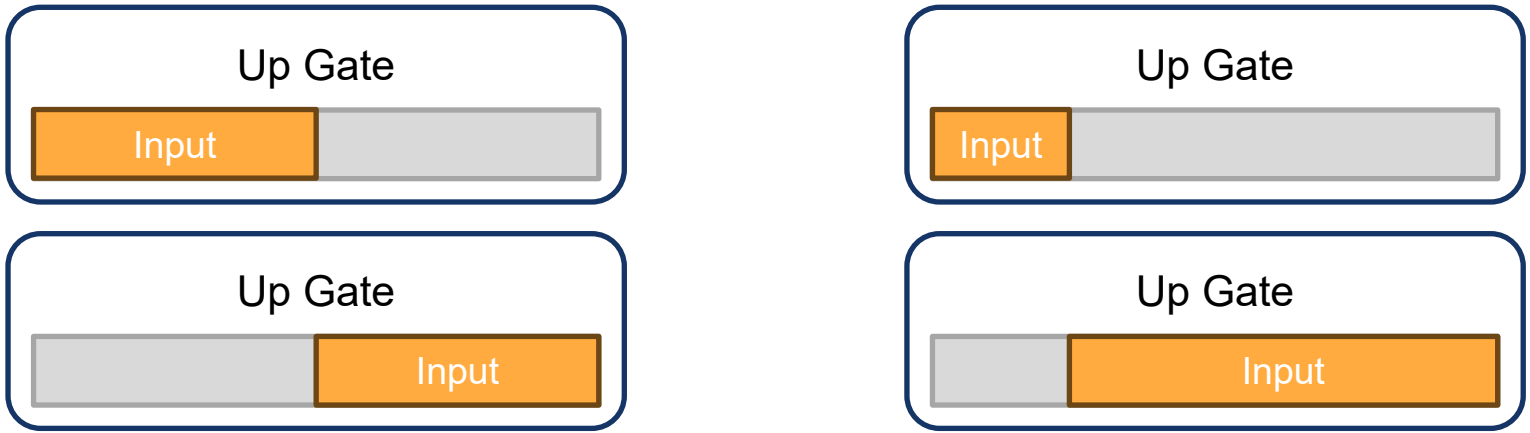
Design Space for overlapping operations

Number of Nanobatch



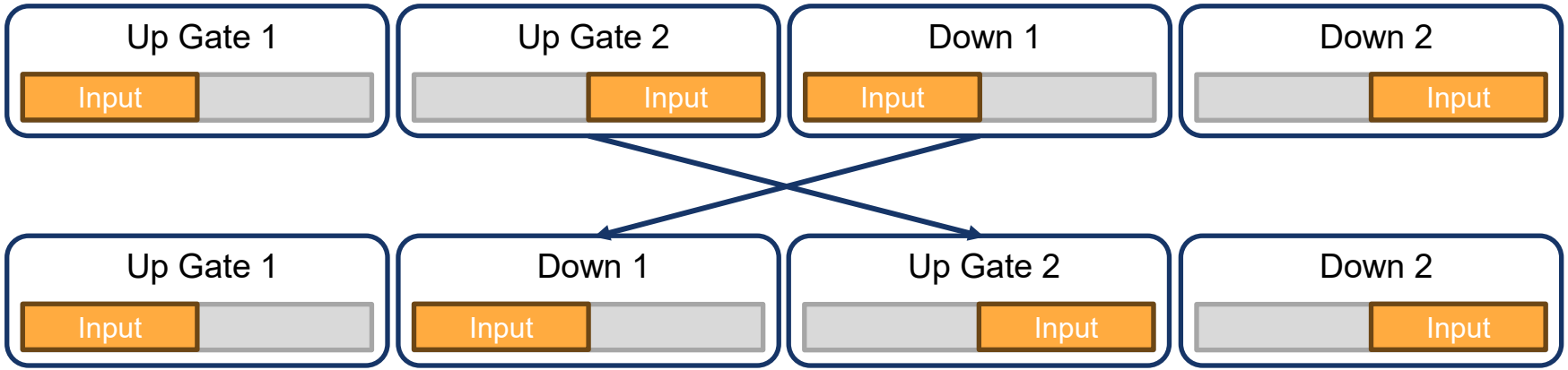
Design Space for overlapping operations

Size of Nanobatch

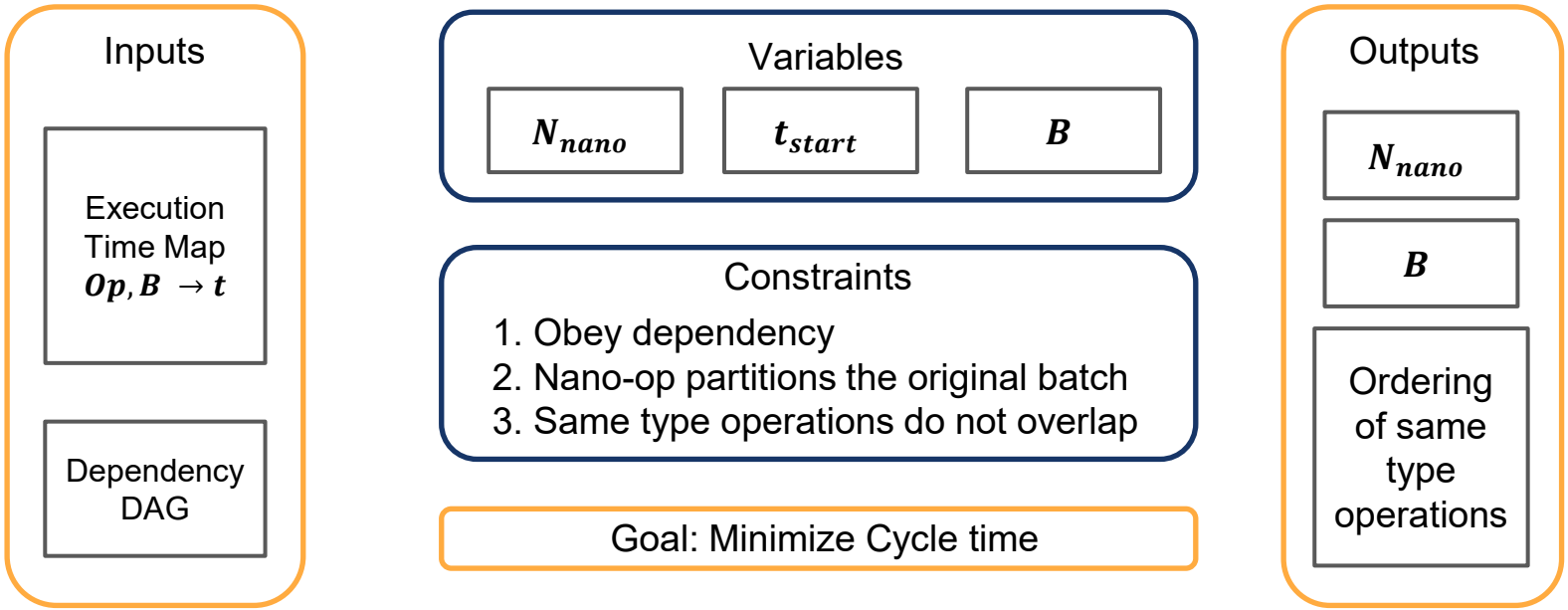


Design Space for overlapping operations

Ordering of Nanobatch



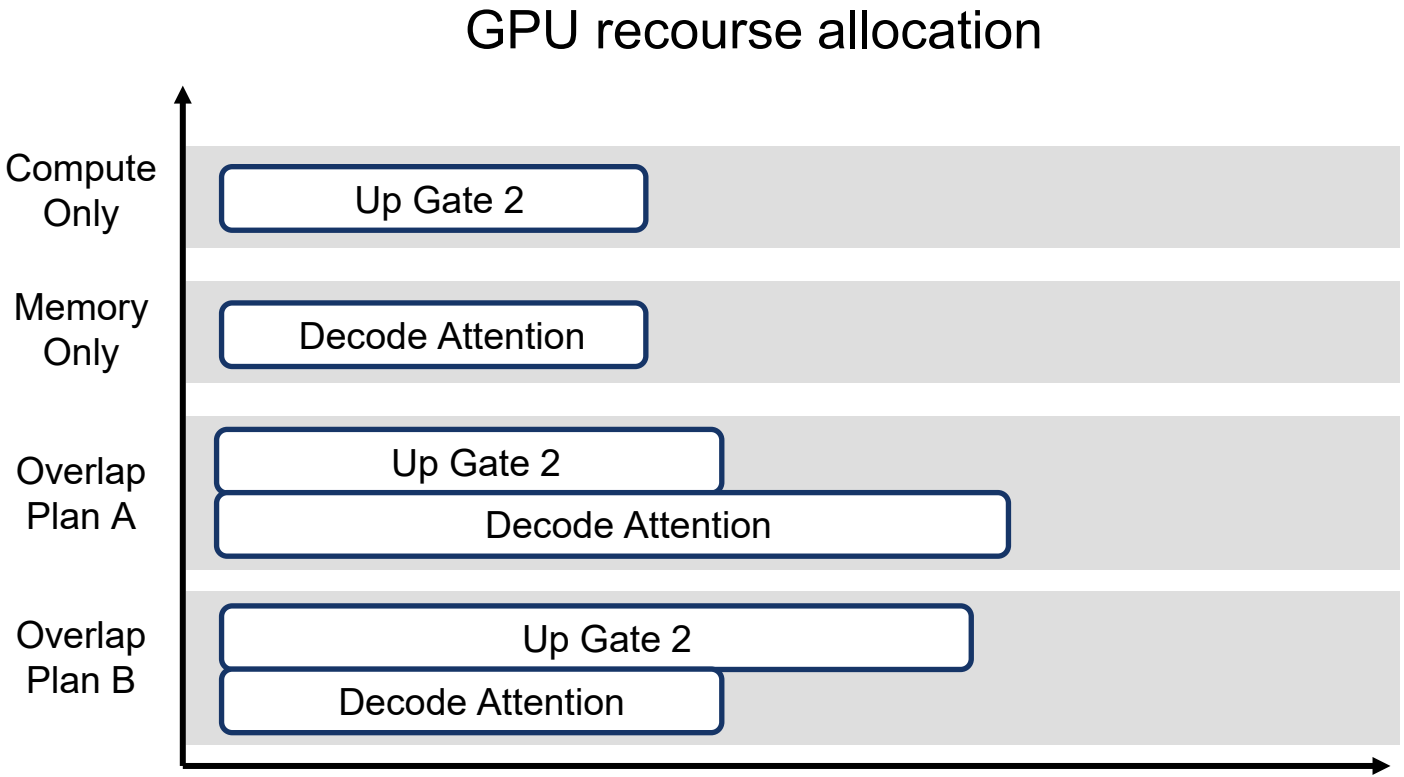
Auto-search Stage 1: Pipeline Search



Simplifications

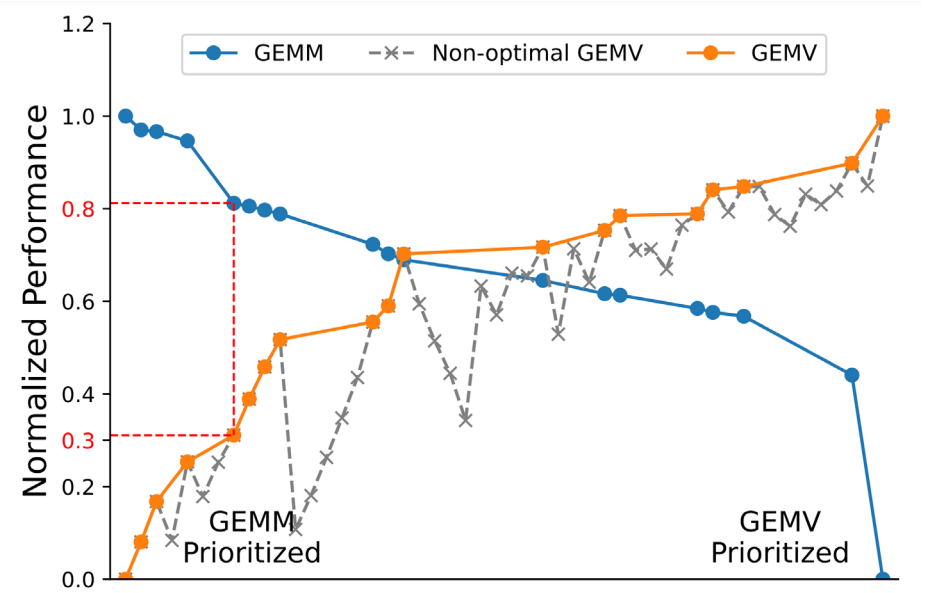
1. Only consider three layers
2. Nanobatch partition are symmetric across layers
3. Operations can overlap perfectly

Design Space for overlapping operations



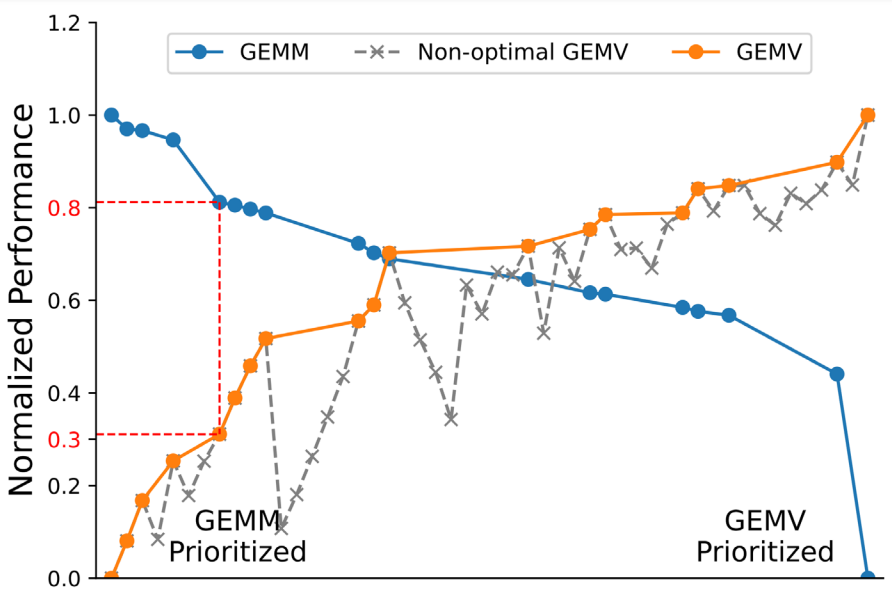
Profiling Interference

- GEMM
  - Tile size
  - Split K
  - Num of warps
  - …
- GEMV
  - Number of thread blocks
  - …



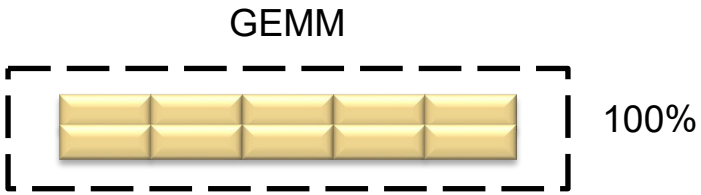
Resource utilization

- We define R as the unified GPU hardware utilization of a kernel
- We use GEMM performance to define the R.
  - E.g. R = 10% means GEMM reaches 10% of Best performance
- GPU have total resource R = 1
- We can get R to performance mappings from profiling

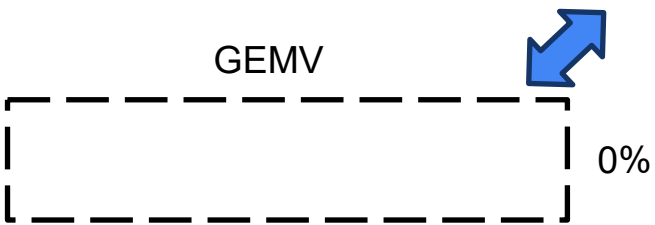


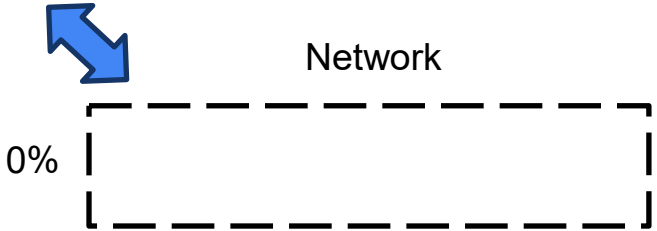
Operations	Resource Utilization (R)						
	0	0.1	0.2	...	0.8	0.9	1
GEMM (by definition)	0	0.1	0.2	...	0.8	0.9	1
GEMV	0	0.2	0.3	...	0.85	0.95	1
Network	0	0.3	0.5	...	0.9	1	1

Resource utilization

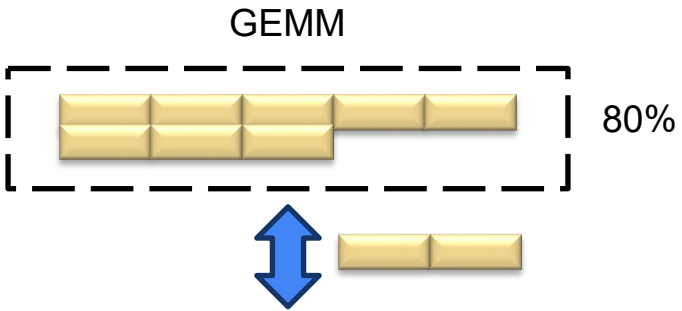


Operations	Resource Utilization (R)						
	0	0.1	0.2	…	0.8	0.9	1
GEMM (by definition)	0	0.1	0.2	…	0.8	0.9	1
GEMV	0	0.2	0.3	…	0.85	0.95	1
Network	0	0.3	0.5	…	0.9	1	1

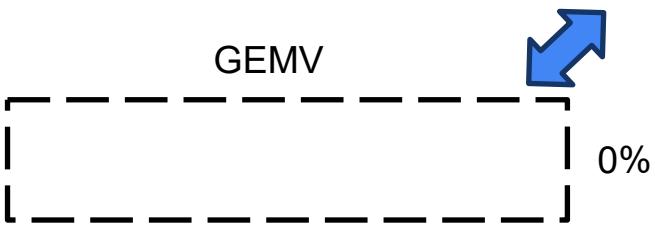


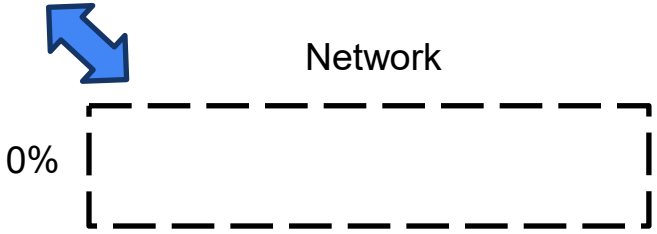


Resource utilization

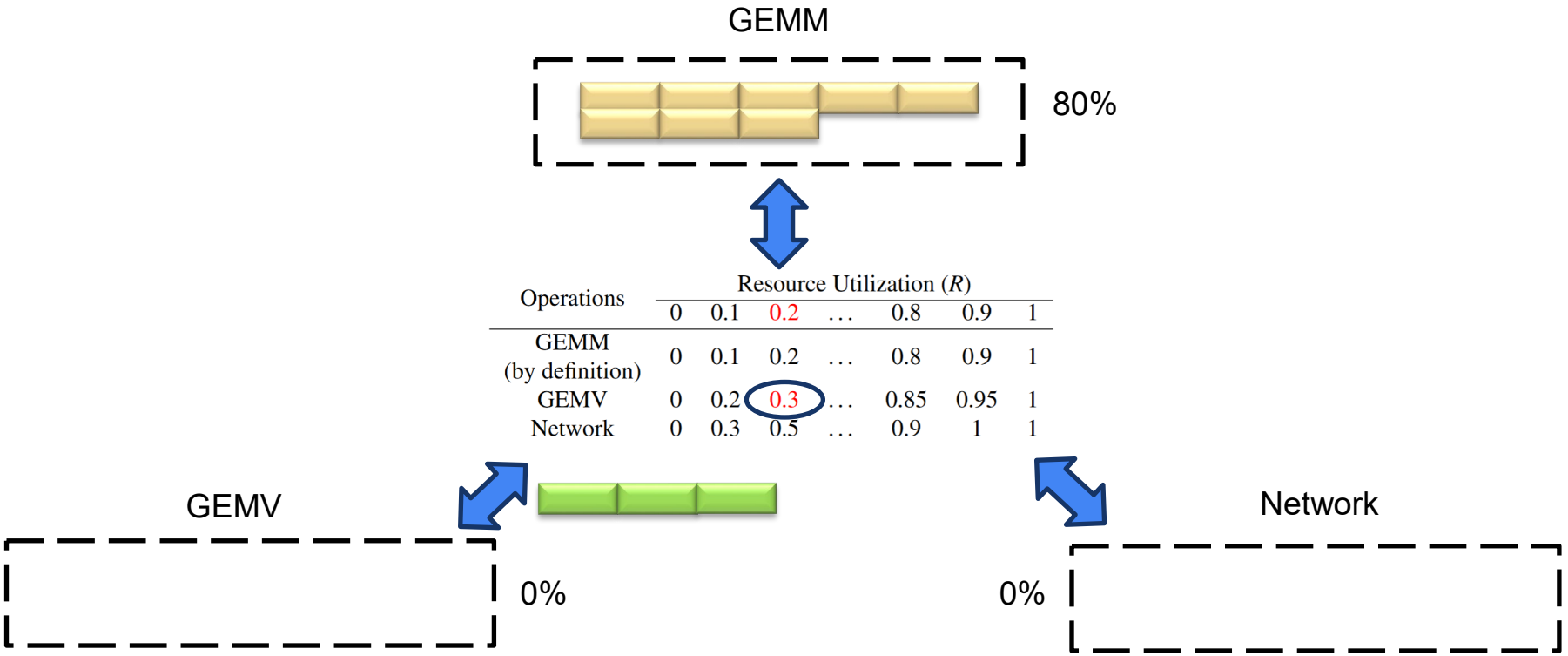


Operations	Resource Utilization (R)						
	0	0.1	0.2	...	0.8	0.9	1
GEMM (by definition)	0	0.1	0.2	...	0.8	0.9	1
GEMV	0	0.2	0.3	...	0.85	0.95	1
Network	0	0.3	0.5	...	0.9	1	1



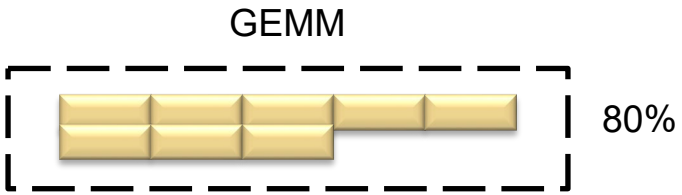


Resource utilization

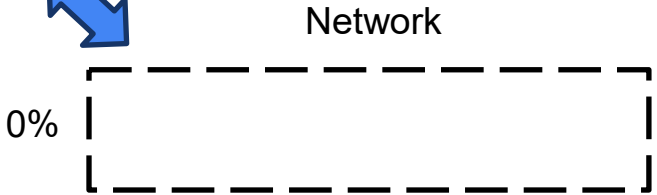


Operations	Resource Utilization (R)						
	0	0.1	0.2	…	0.8	0.9	1
GEMM (by definition)	0	0.1	0.2	…	0.8	0.9	1
GEMV	0	0.2	0.3	…	0.85	0.95	1
Network	0	0.3	0.5	…	0.9	1	1

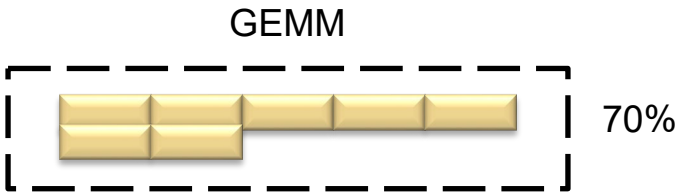
Resource utilization



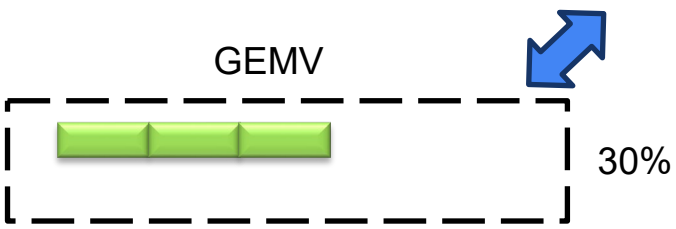
Operations	Resource Utilization (R)						
	0	0.1	0.2	…	0.8	0.9	1
GEMM (by definition)	0	0.1	0.2	…	0.8	0.9	1
GEMV	0	0.2	0.3	…	0.85	0.95	1
Network	0	0.3	0.5	…	0.9	1	1

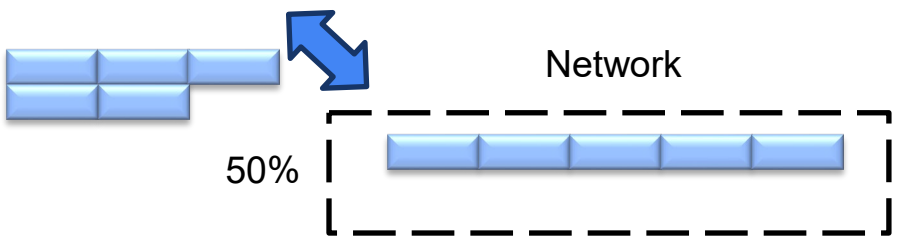


Resource utilization

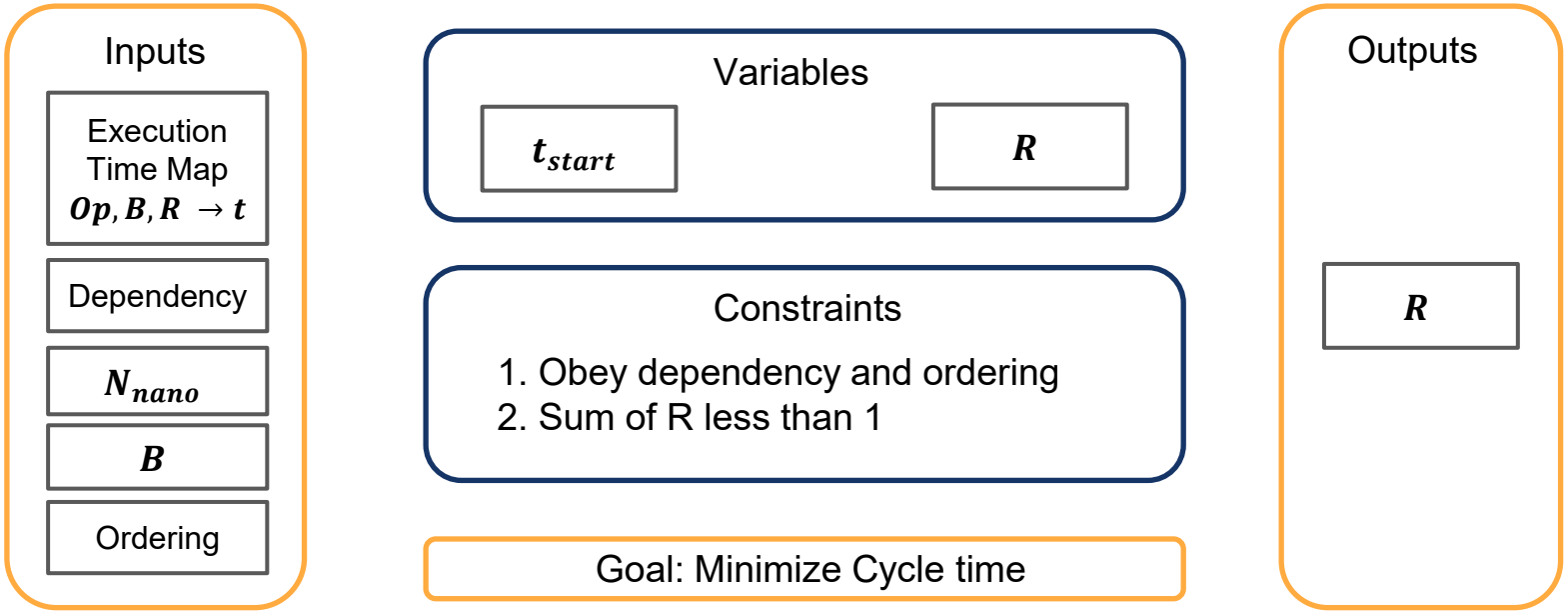


Operations	Resource Utilization (R)						
	0	0.1	0.2	…	0.8	0.9	1
GEMM (by definition)	0	0.1	0.2	…	0.8	0.9	1
GEMV	0	0.2	0.3	…	0.85	0.95	1
Network	0	0.3	0.5	…	0.9	1	1





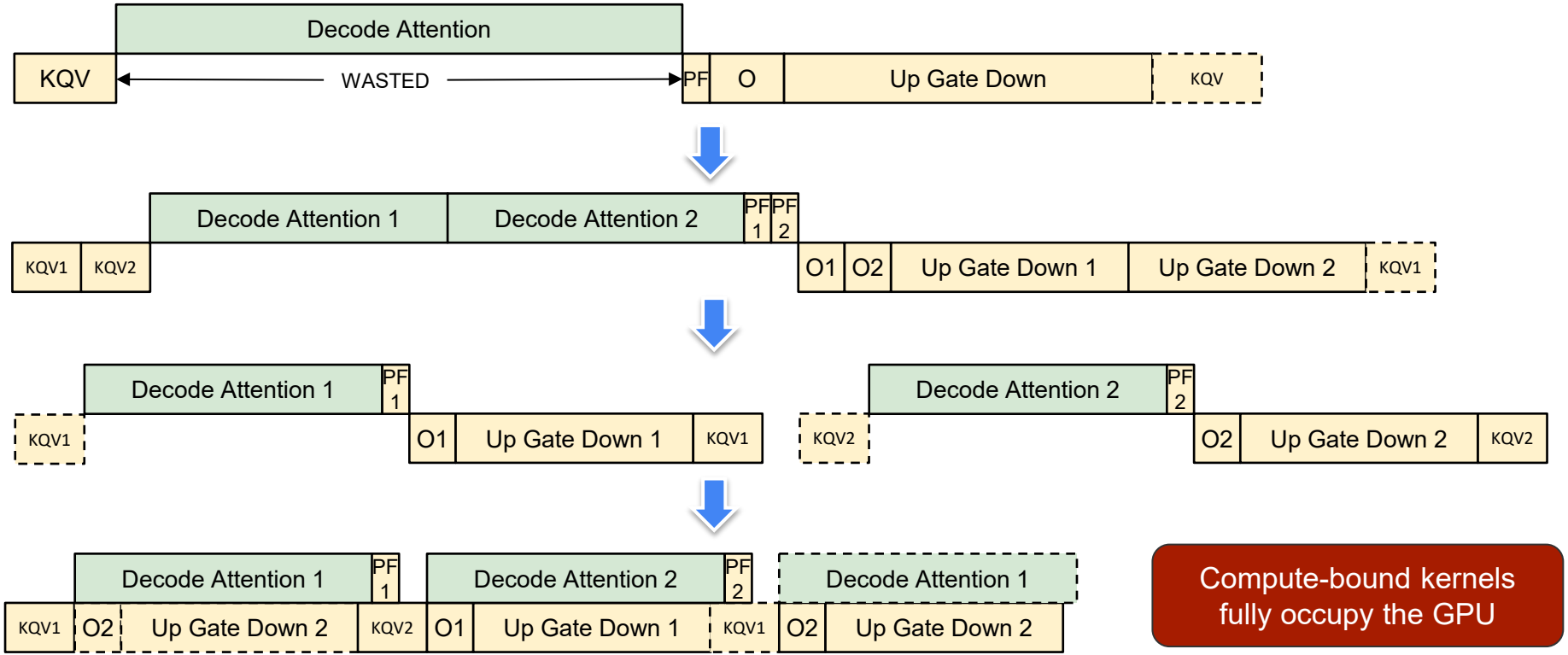
Auto-search Stage 2: Resource allocation



Simplifications

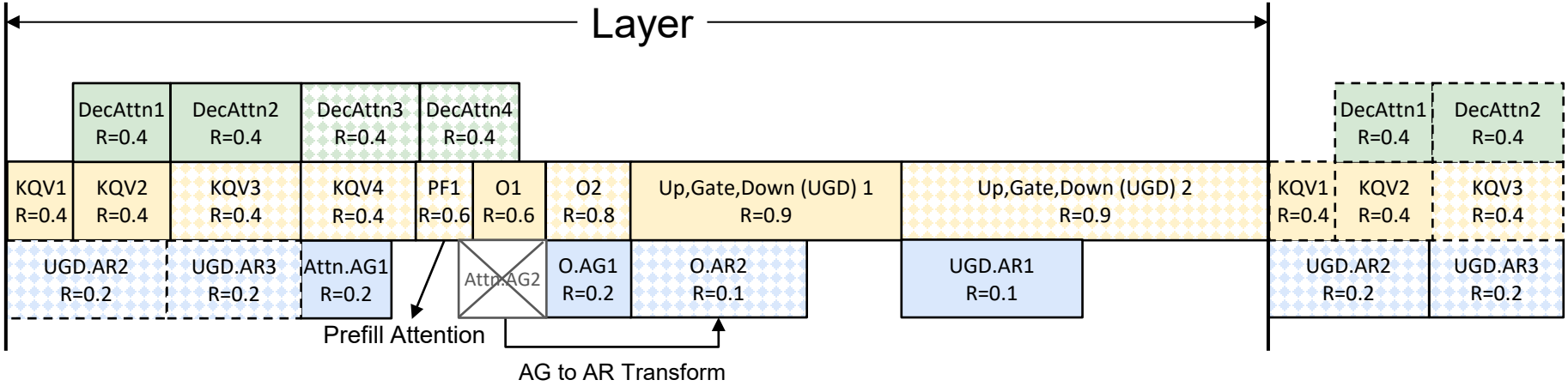
1. Only consider three layers 2. R is symmetric across layers

8B Pipeline Design



Compute-bound kernels fully occupy the GPU

70B Pipeline Design



NanoFlow

What is the optimal LLM serving throughput?

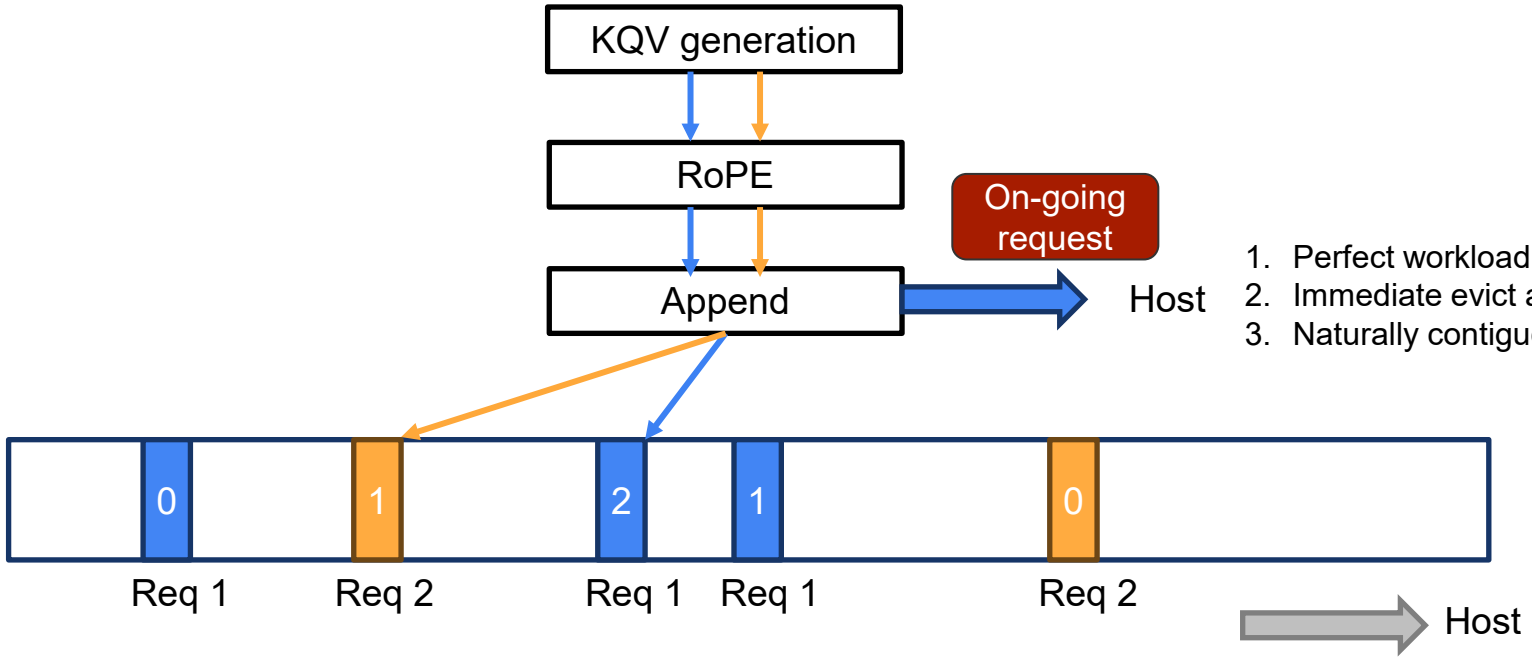
- Operation Classification
- Resource Bottleneck analysis

How to approach optimal LLM serving?

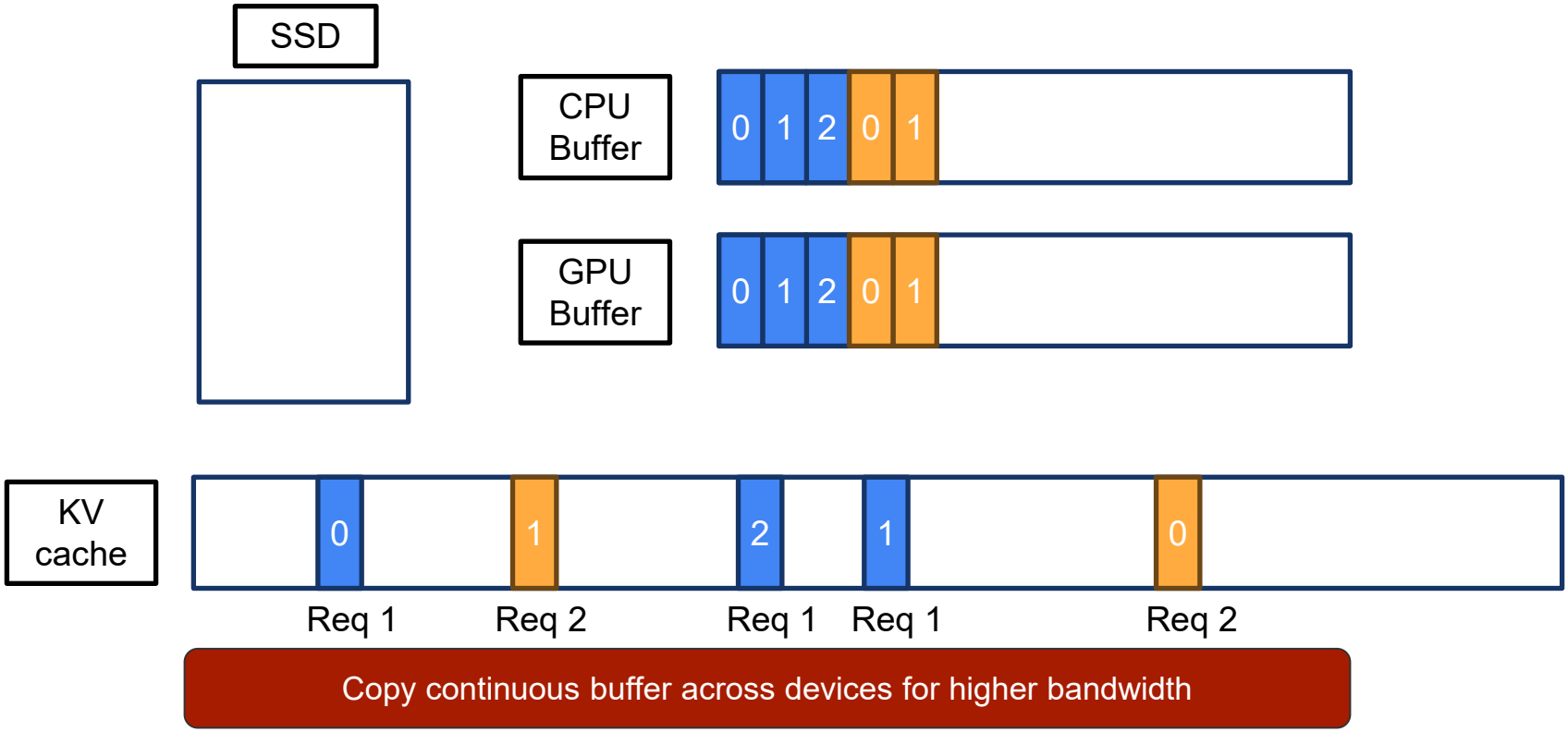
- Nanobatch
- AutoSearch
- Async Scheduling
- KV cache management

How does Nanoflow compare with existing frameworks?

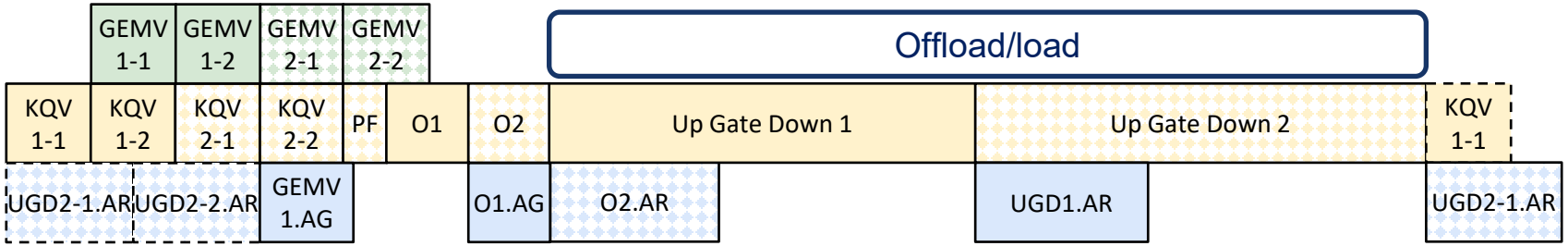
Offloading requests



Loading requests

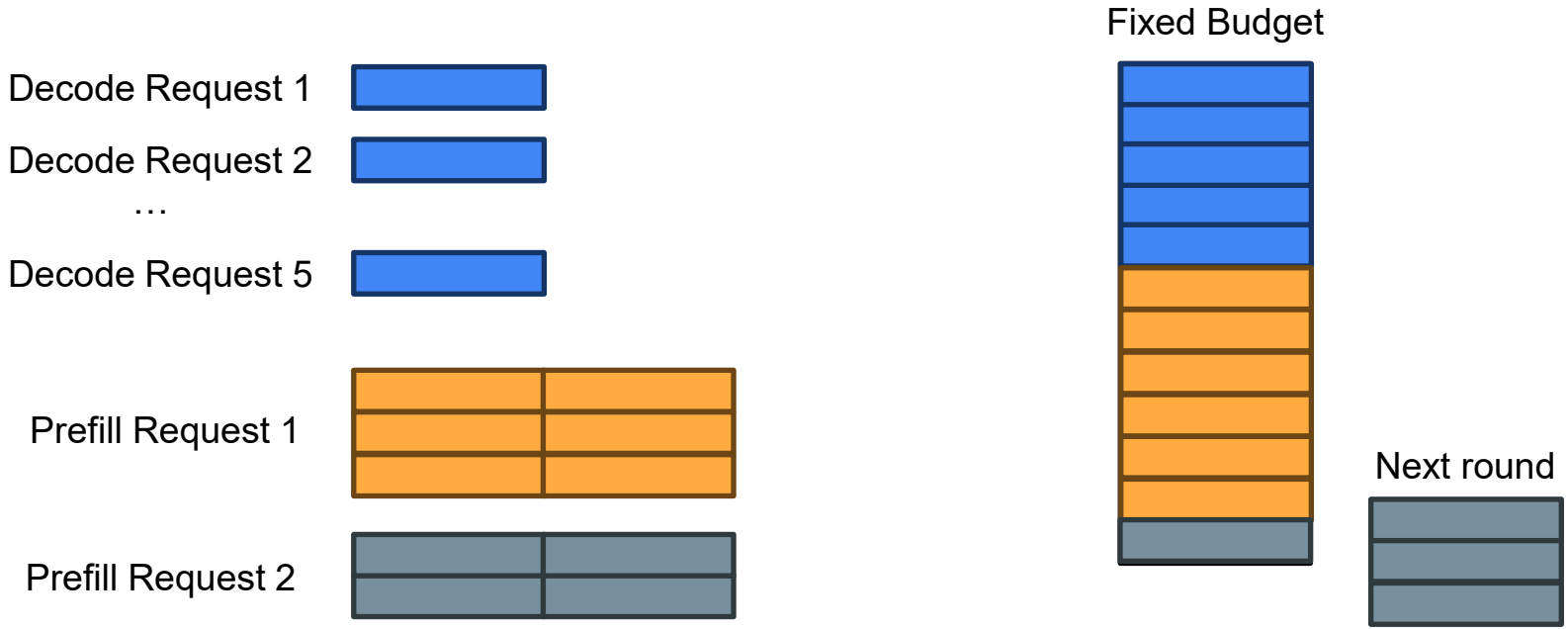


Integrating Loading and Offloading to Pipeline



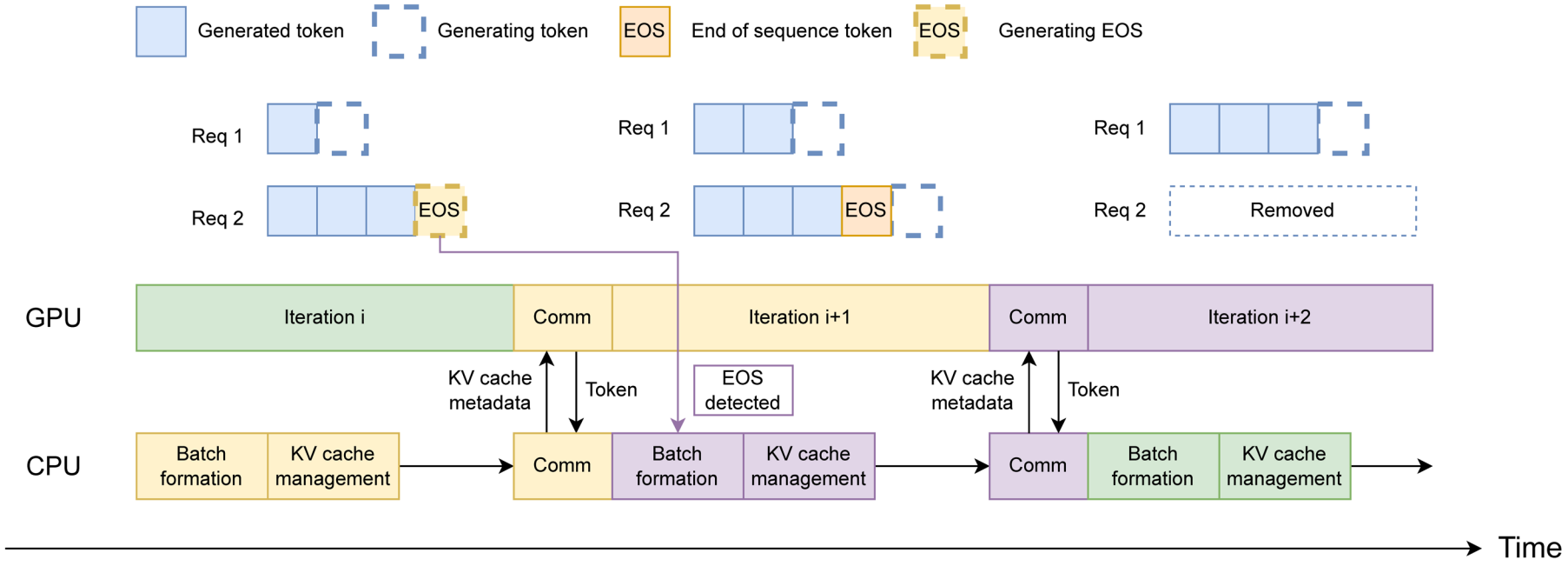
Overlap offloading with pipeline execution to reduce overhead

Batch Formation



Nanoflow form constant batch for consistent performance

Asynchronous scheduling



Batch formation and KV cache management are moved out from critical path

NanoFlow

What is the optimal LLM serving throughput?

- Operation Classification
- Resource Bottleneck analysis

How to approach optimal LLM serving?

- Nanobatch
- AutoSearch
- Async Scheduling
- KV cache management

How does Nanoflow compare with existing frameworks?

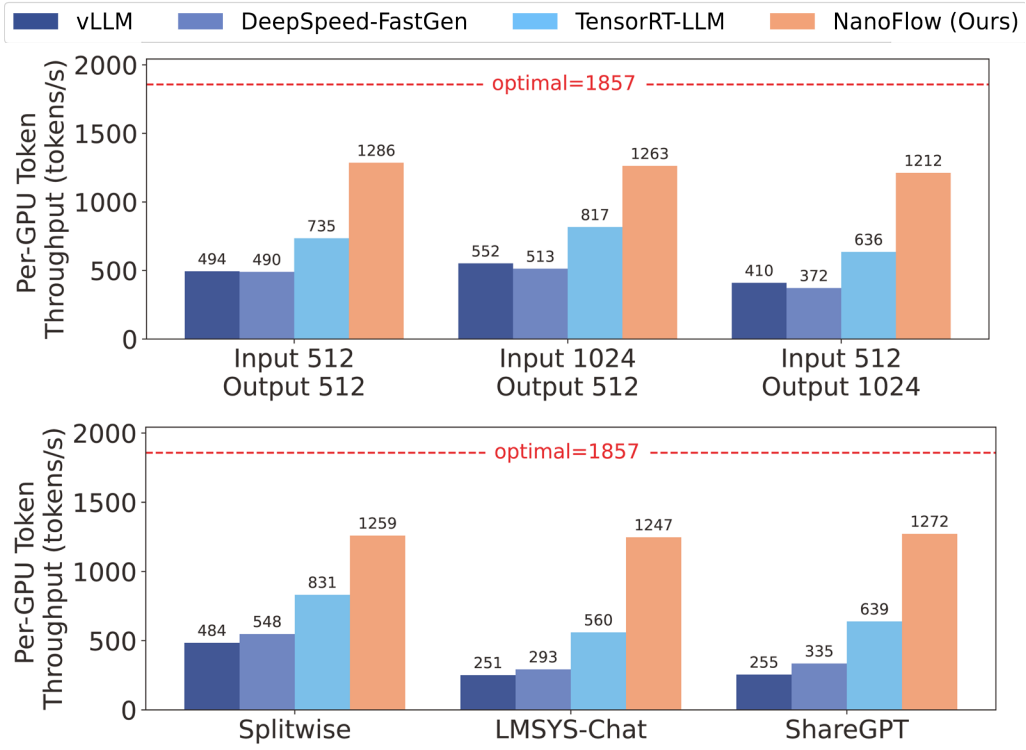
Evaluation: end-to-end offline throughput

Dataset	Avg. Input (Std)	Avg. Output (Std)
Splitwise [34]	1155 (1109)	211 (163)
LMSYS-Chat-1M [52]	102 (169)	222 (210)
ShareGPT [1]	246 (547)	322 (244)

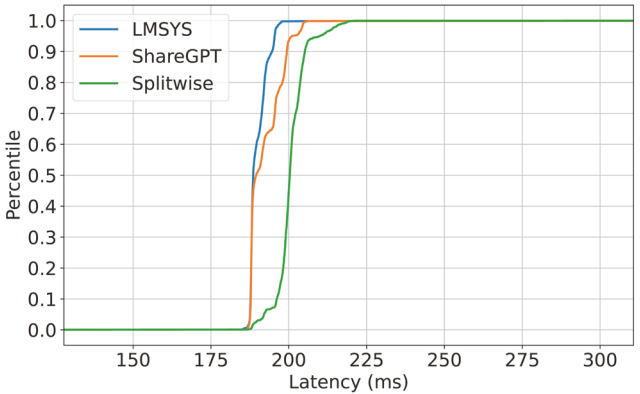
1.73x const length throughput gain

1.91x dataset throughput gain

68.5% of optimal throughput



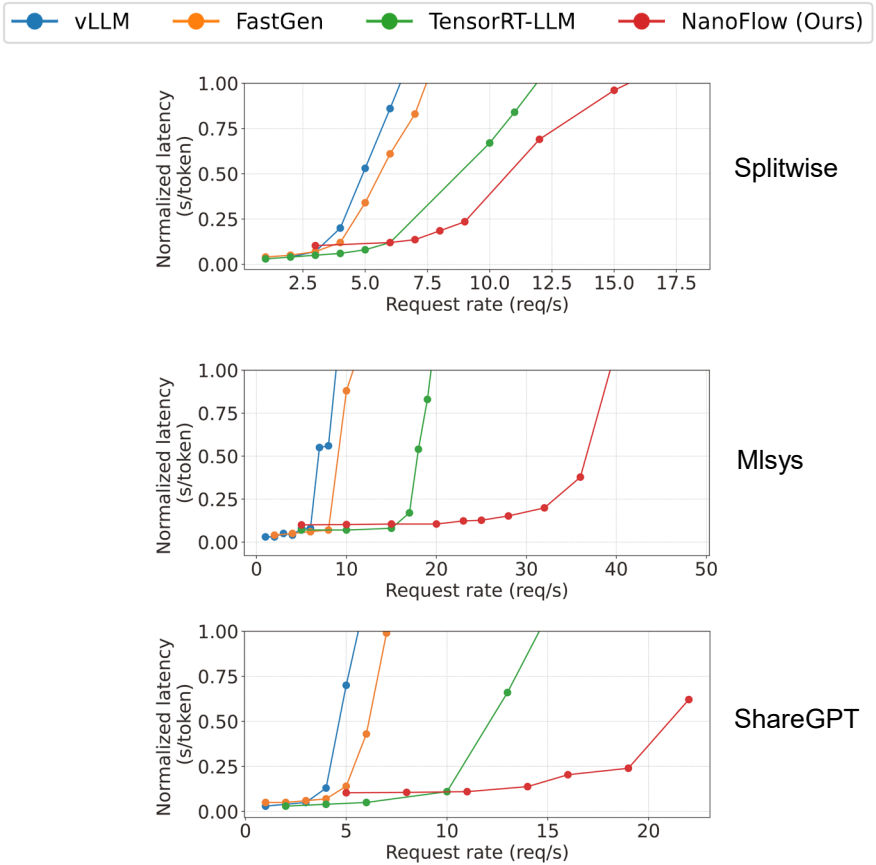
Evaluation: online latency



Low tail latency

Similar latency at low request rate

Process 1.64x requests at 200ms SLO

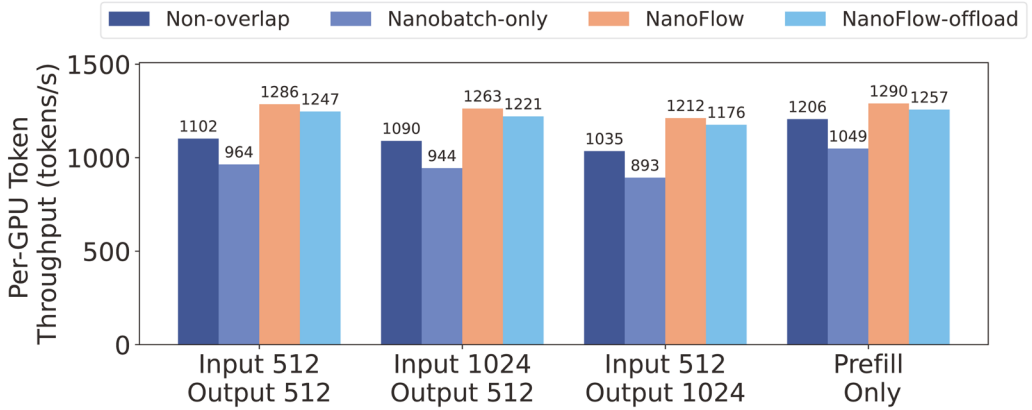


Evaluation - Ablation study

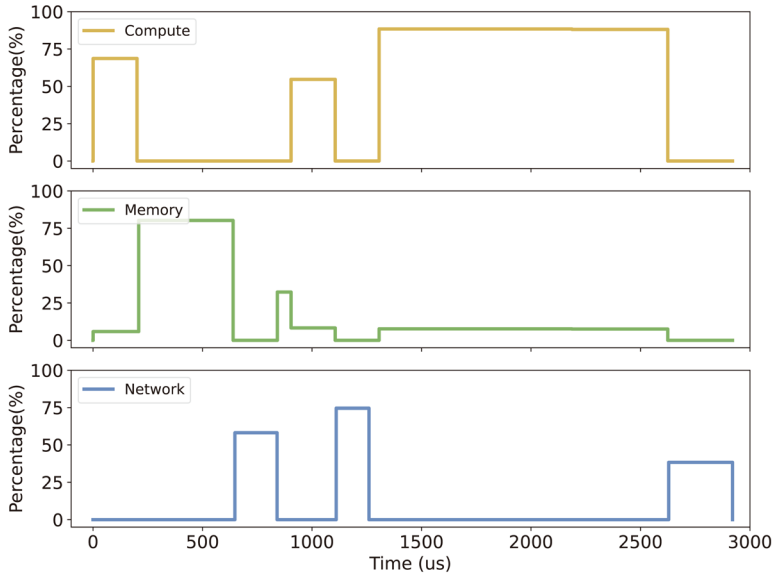
Overlapping network 1.07x speedup

Overlapping memory 1.10x speedup

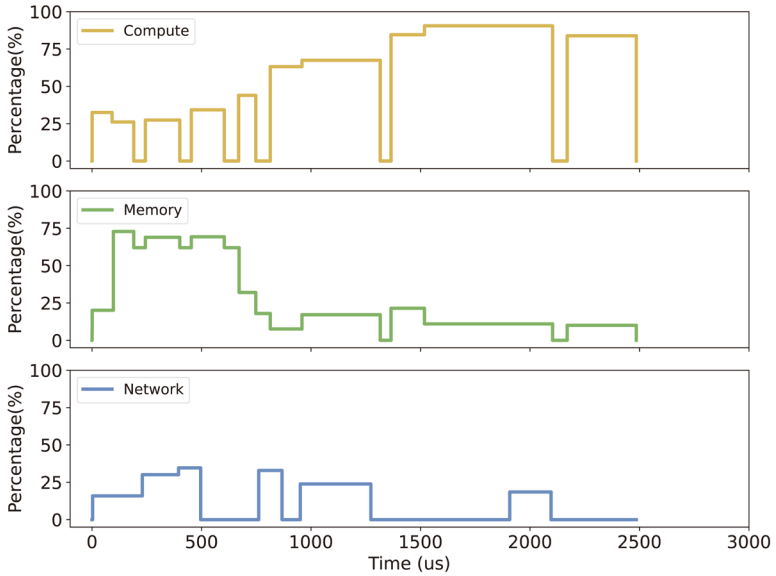
Overall, 1.17x speedup



Evaluation - Resource usage



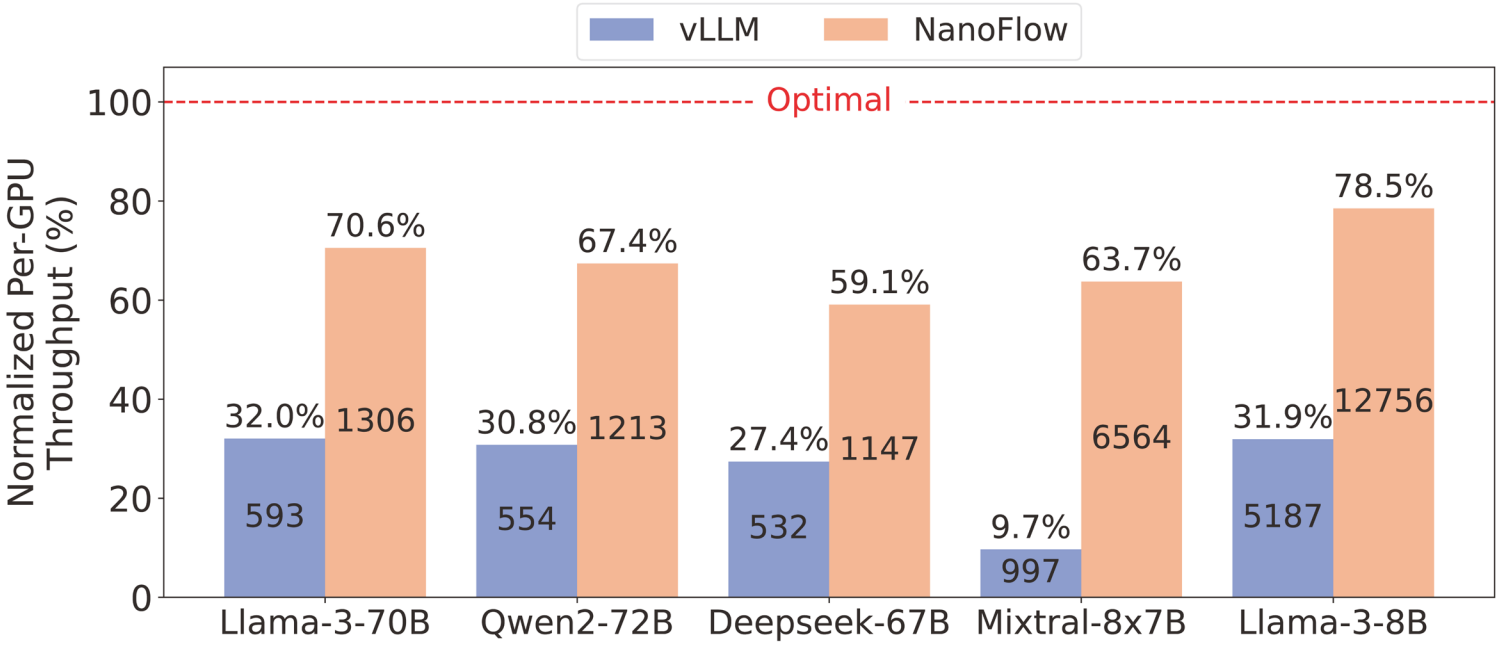
Non-overlap baseline



Nanoflow

Nanoflow utilize multiple resources concurrently, achieving higher average compute usage.

Evaluation - Porting Nanoflow to other models



Porting Nanoflow to other models demonstrates 59.1% to 78.5% of optimal throughput

NanoFlow

Paper

Github

Post

Nanobatch pipeline design

Execution unit scheduling

1.91x throughput gain

78.5% optimal throughput