

THE UNIVERSITY of EDINBURGH

Tencent 腾讯

BatchGen:

An Architecture for Scalable & Efficient Batch Inference

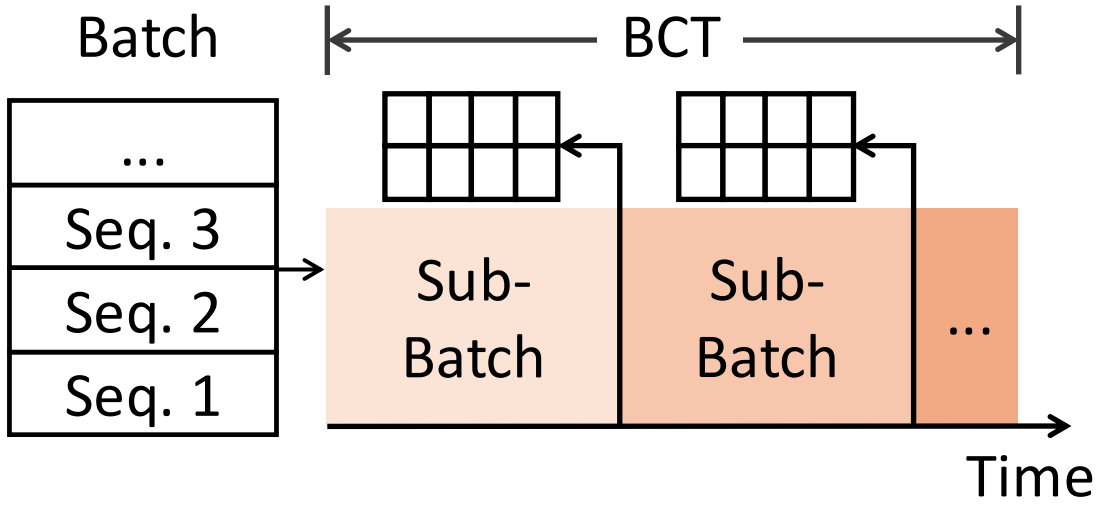
Tairan Xu (tairan.xu@ed.ac.uk)

Tairan Xu*, Leyang Xue*, Zhan Lu*,
Jinfu Deng, Hongyang Xiao, Yinsicheng Jiang, Congjie He, Matej Sandor, Le Xu, Luo Mai

OSDI'26 · University of Edinburgh & Tencent

Batch Inference Definition

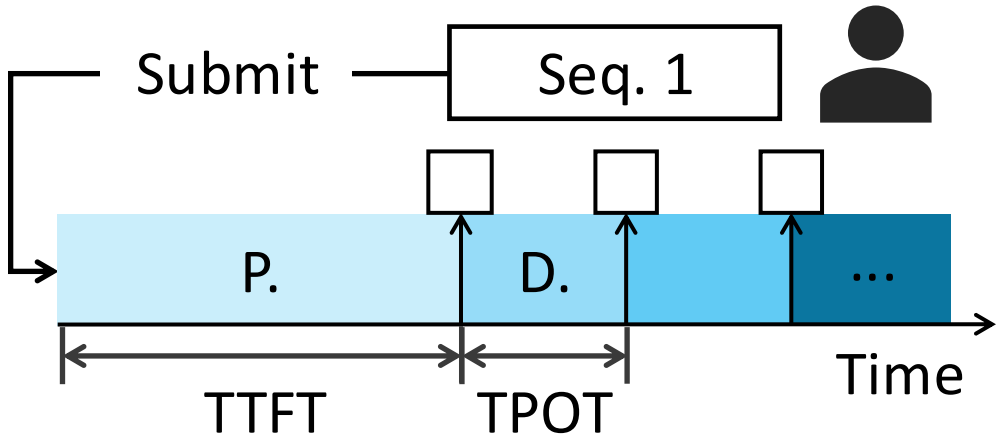
Batch Inference



Client side only cares

Batch Completion Time (BCT)

Interactive Inference



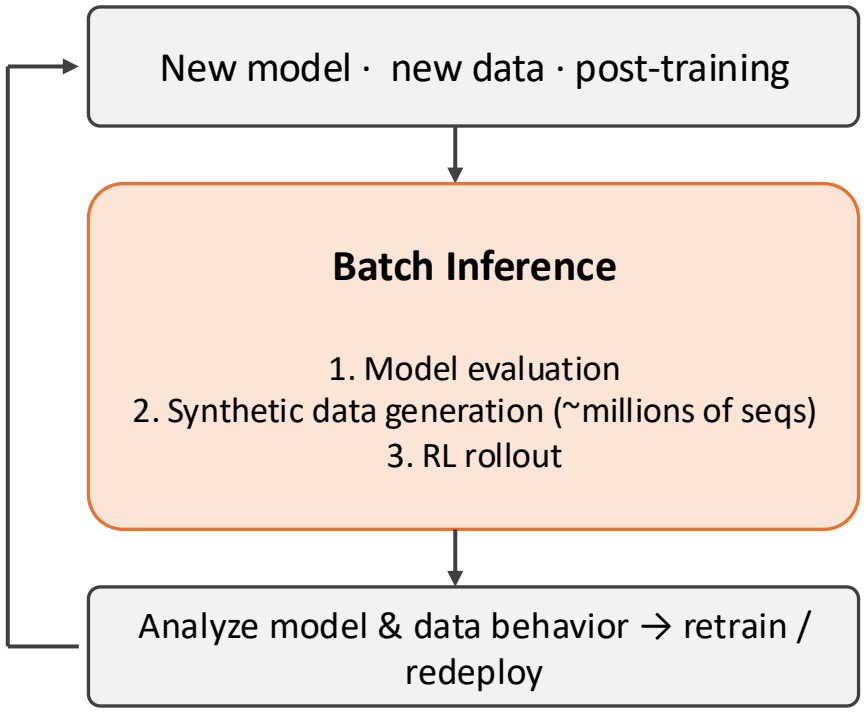
Client side cares the response latency

Time to First Token (TTFT)

Time per Output Token (TPOT)

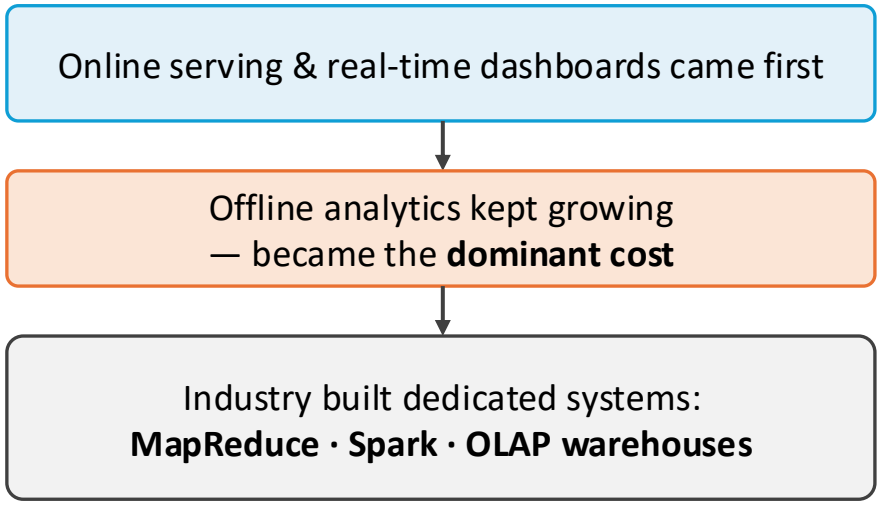
Why It Matters?

Recurring by Nature



…re-runs at every release and every data refresh

Analogy: The Big-Data Era



Batch inference is at the same inflection point — it deserves its own architecture.

Even online workloads are turning batch!

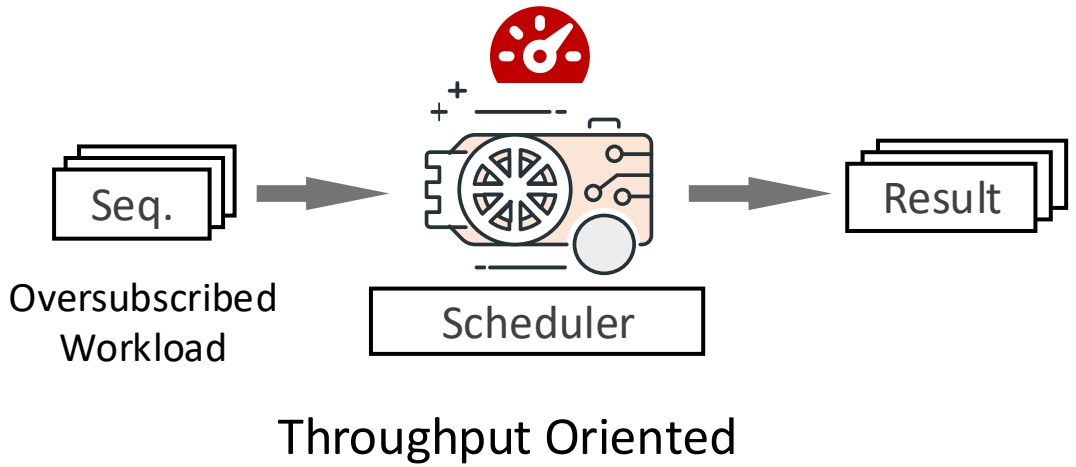
Test-time compute:
~1,000 trajectories per query
— AlphaEvolve[1]

Already a separate product tier: OpenAI & Anthropic batch APIs are priced at **50% off**[2]

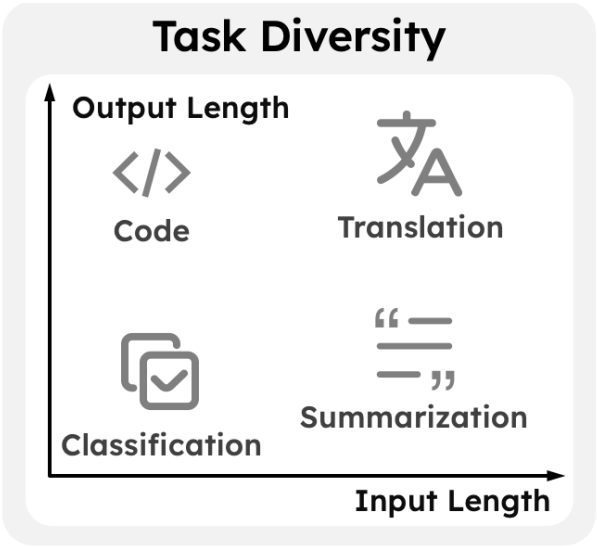
1. Novikov et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv:2506.13131, 2025.
2. OpenAI Batch API and Anthropic Message Batches API public pricing, as of July 2026.

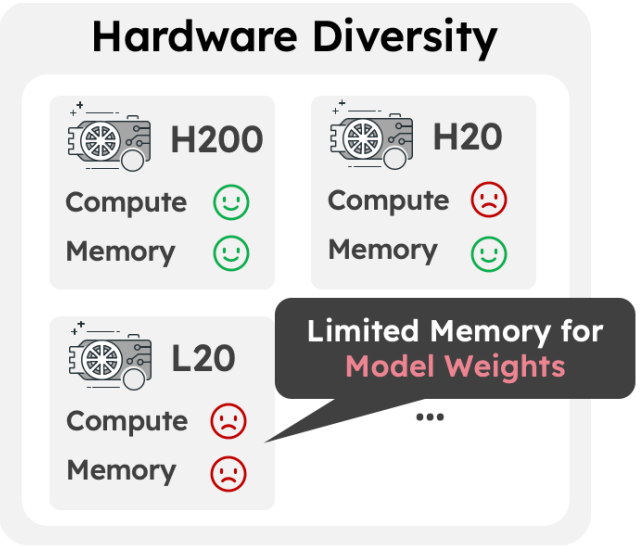
Design Philosophy of Batch Inference

Reduce Batch Completion Time
- Throughput Oriented execution



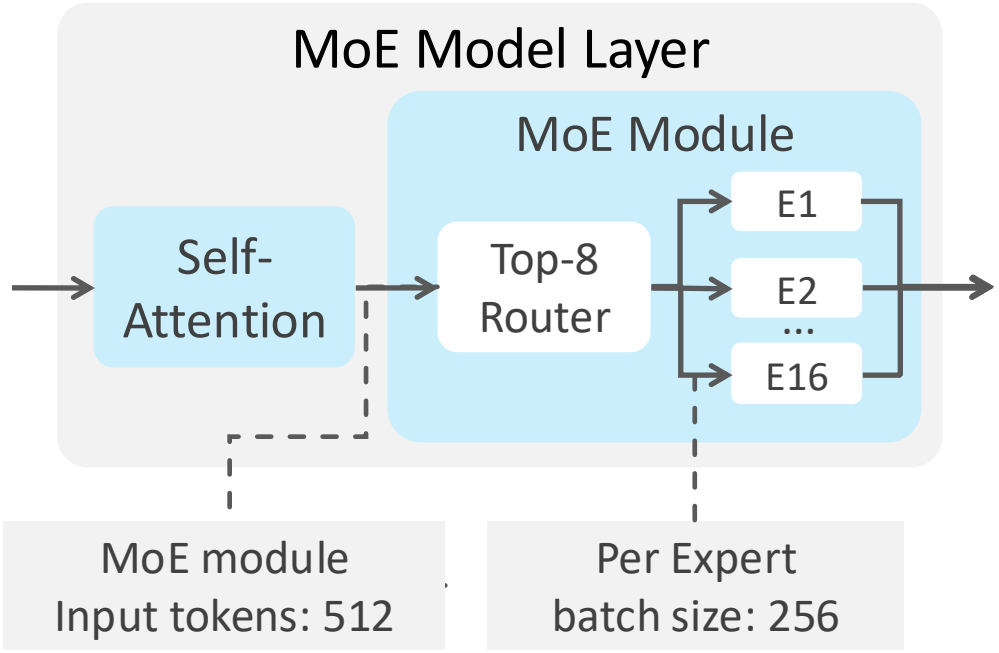
Flexibility Across Tasks and Hardware
- Task adaptability
- Hardware adaptability





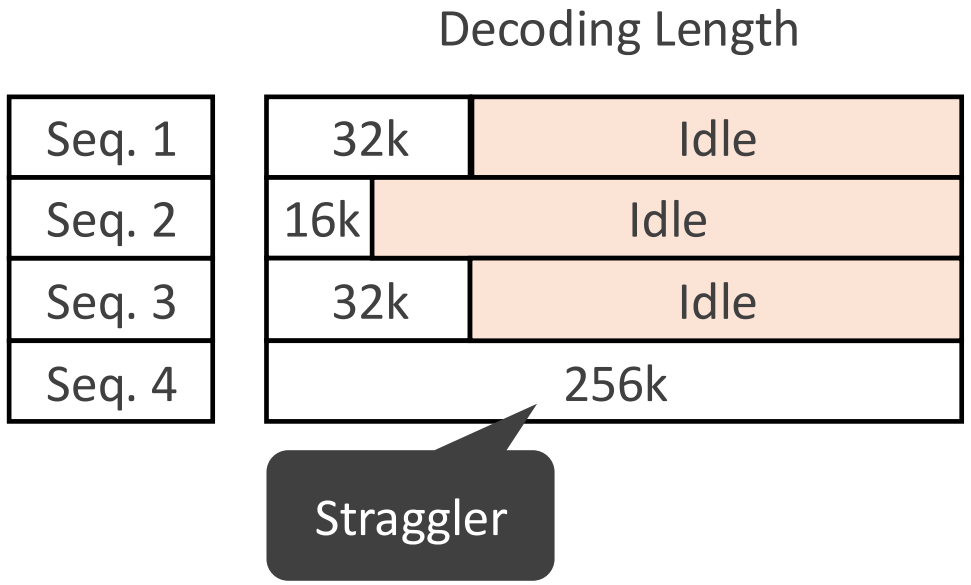
Issue 1: GPU Underutilization

Model Sparsity



Two nodes H800, DeepSeek-R1
Peak FP8 MFU: 90%.
Reality: under 50%

Long Tail



Issue 2: Task and Hardware Diversity

1. Shifting workload and fluctuating resources.

P:D Ratio	1:7	2:6	3:5	4:4	5:3	6:2	7:1
PD Disagg. (min)	137.9	67.4	49.0	38.6	40.1	55.6	112.0

Deepseek-R1, 8K-2K, 128 H20

2. Heterogeneous hardware.
   - Often OOM when switching GPU types (H200 -> L20)

1. Zhong et al. DistServe: Disaggregating Prefill and Decoding for Goodput-Optimized Large Language Model Serving. OSDI 2024.
2. Zhu et al. MegaScale-Infer: Efficient Mixture-of-Experts Model Serving with Disaggregated Expert Parallelism. SIGCOMM 2025.

Opportunities: Making Sequences as Coroutine

- Per sequence latency is less relevant.
- A sequence can be modelled as “coroutine”
- yielded, re-organized and resumed as long as throughput benefits.

	Interactive Inference	**Disaggregated Inference**	**Sequence Coroutine (Ours)**
Scheduling plan	Static	Static	**Adaptive**
Scheduling granularity	Coarse-grained, fixed	Coarse-grained, fixed	**Fine-grained, flexible**
Operational cost	Medium	High	**Low**
Throughput	Medium	Medium	**High**

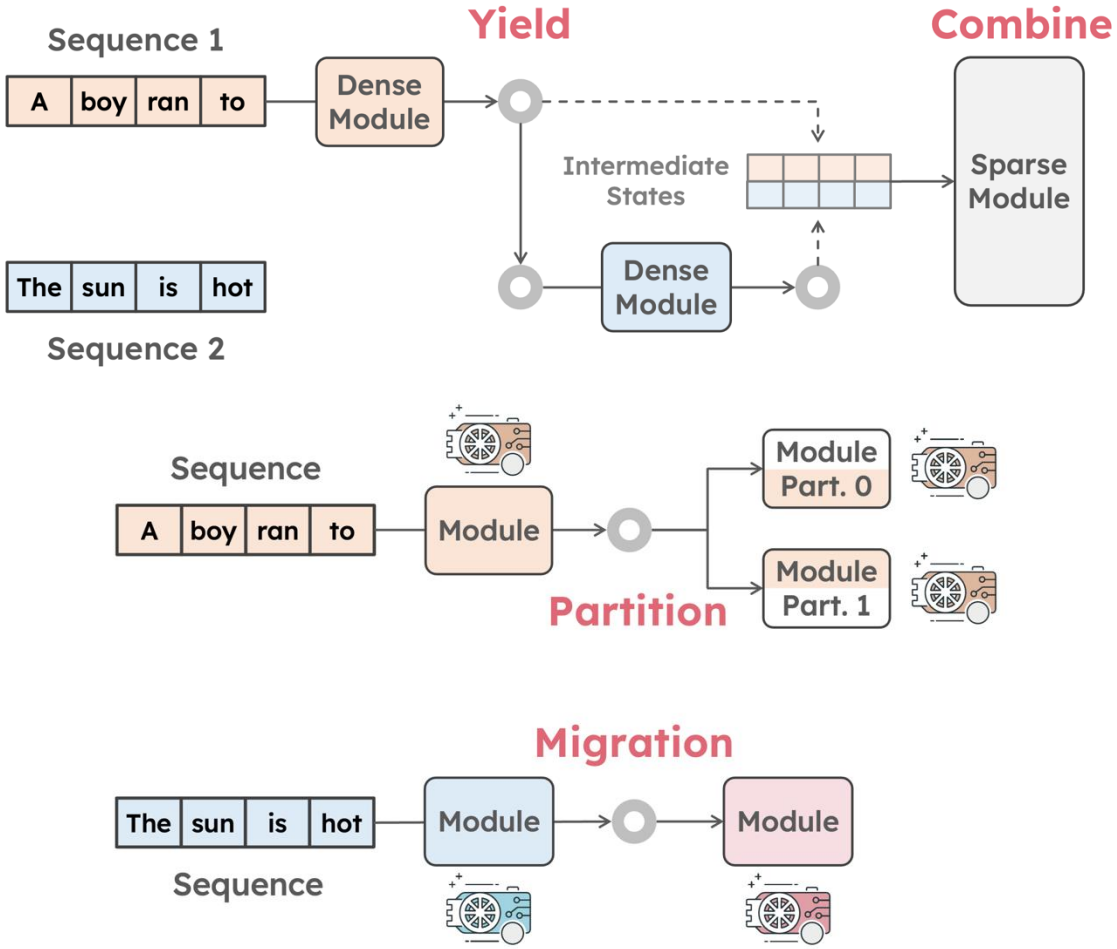
How to enable the dynamic scheduling

Sequence Coroutine Model

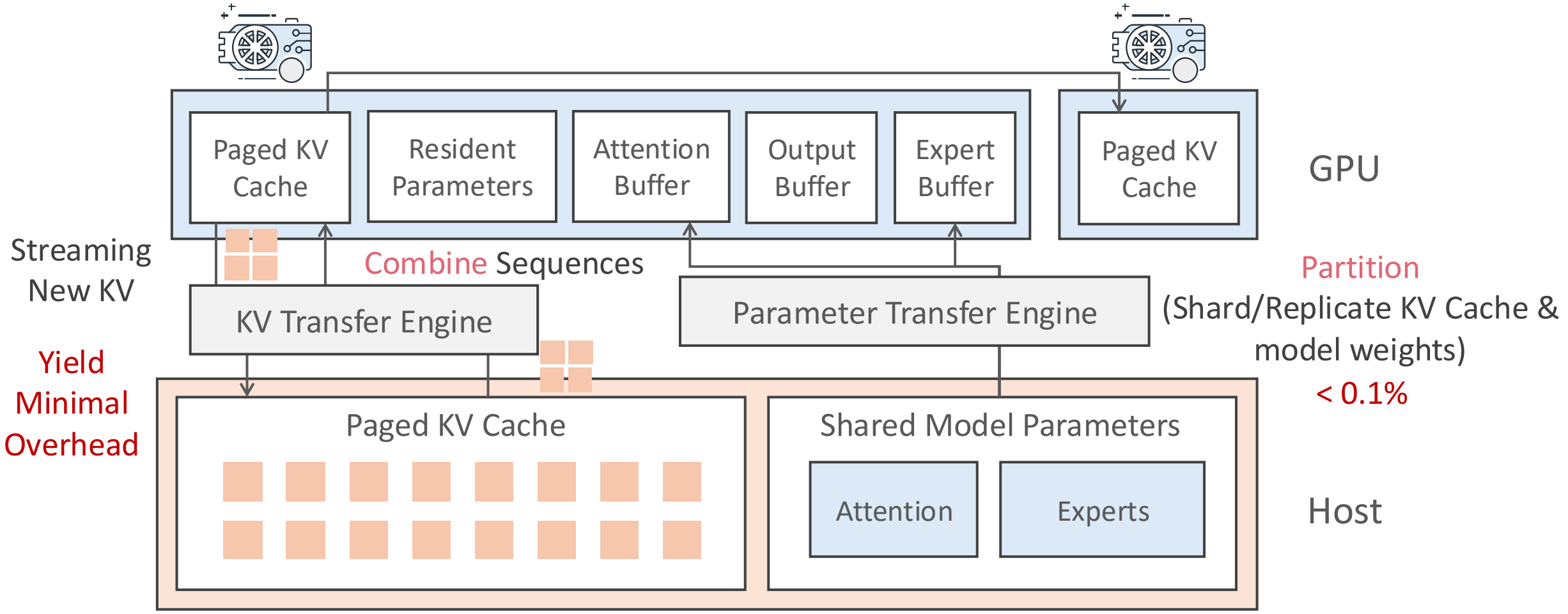
```python
class SequenceCompute:
  seq_id: int; max_out: int; max_in: int; tokens: str
  phase: Literal["prefill", "decoding"]
  status: Literal["init", "active", "inactive", "done"]
  state: object; output: object

  async def yield(...): # Scheduler takes control
  async def combine(...): # Multiple states
  async def partition(...): # Model parallelism
  async def migrate(...): # Physical location
  async def run(...): # Call module forward

  def callback(...): # Custom local state handling
```

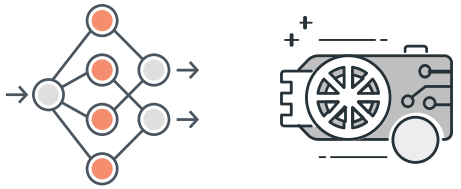


System Runtime to Manage the Coroutine Overhead



Scheduling Policy

Model Hardware

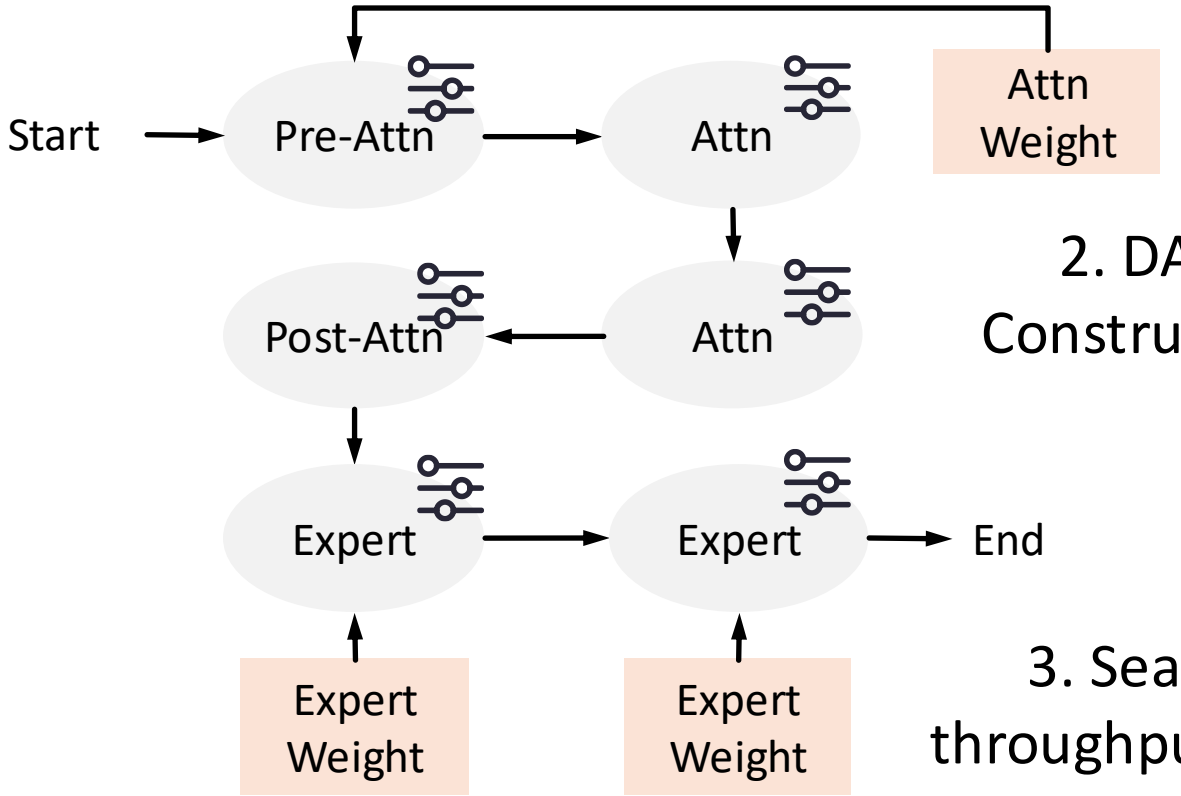


Profile

Metric	Value
Attention	10ms
PCIe-BW	50GB/s
…	…

1. Offline Profiling

Intra-model yield/combine behaviors

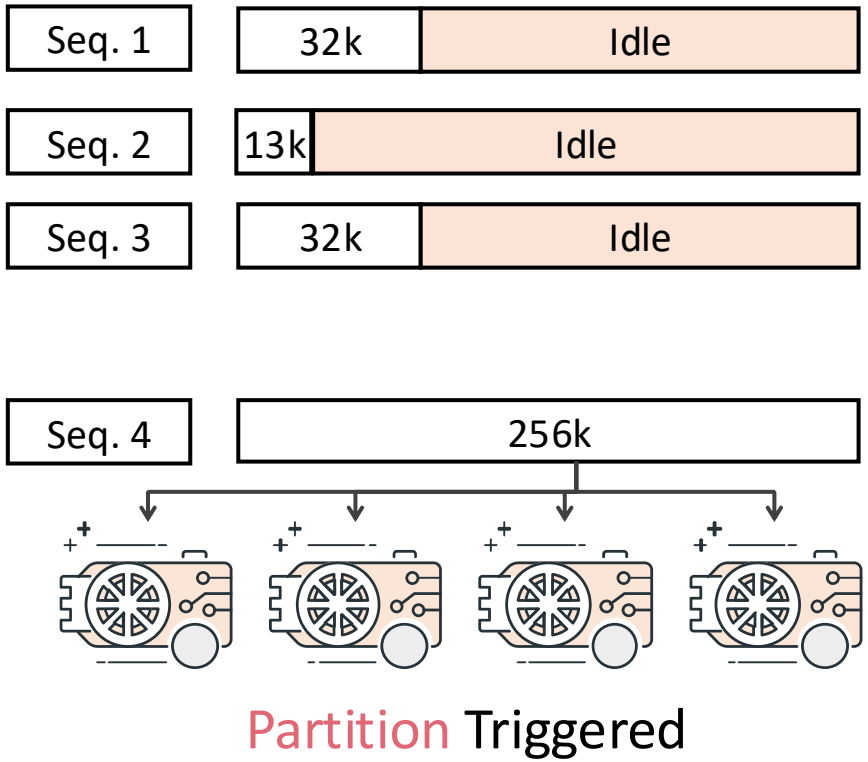


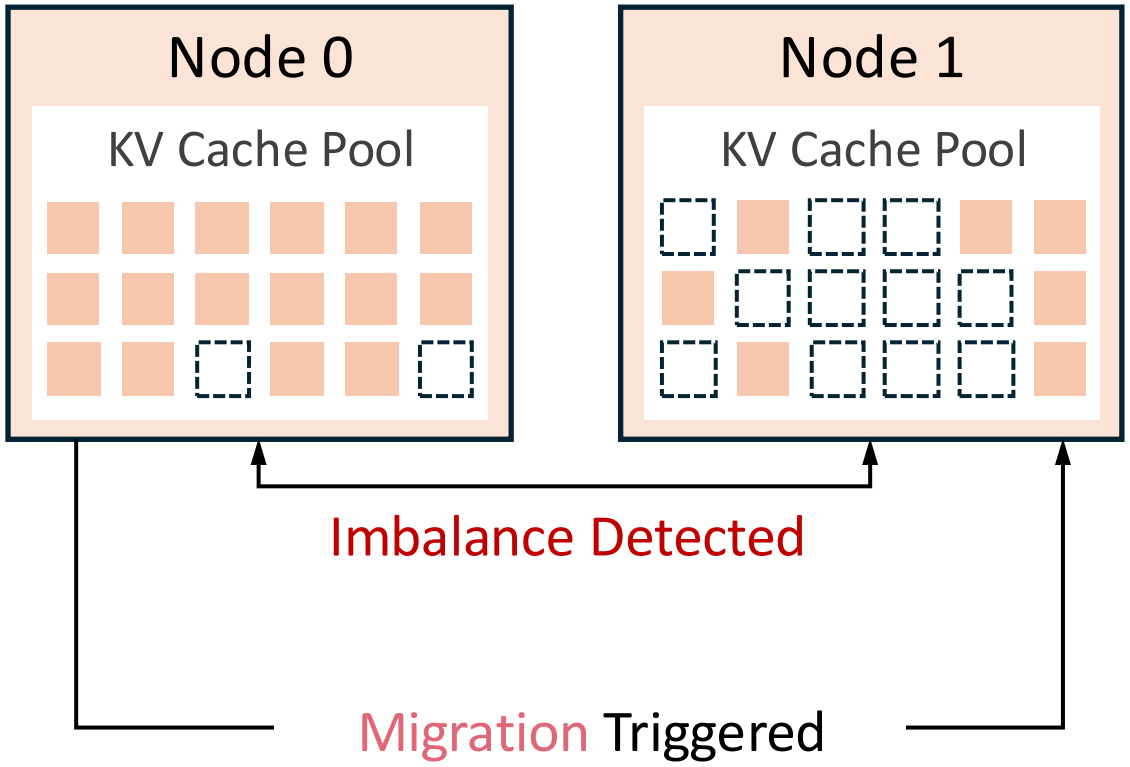
2. DAG Construction

3. Searching throughput optimal configs

Scheduling Policy

Dynamic Sequence Management





End to end results - Throughput

Prefill heavy task, e.g. data curation

	1×8 H20			2×8 H20					
Framework	64K-1	128K-1	255K-1	64K-1	128K-1	255K-1	64K-1K	128K-1K	255K-1K
SGLang	233.1	637.5	OOM	106.2	528.8	1912.6	213.5	559.3	1829.5
SGLang-Opt	233.1	605.6	OOM	106.2	304.8	1202.9	182.9	456.0	1268.6
BatchGen	**160.2**	**455.9**	**1497.8**	**86.7**	**234.3**	**748.4**	**146.8**	**303.6**	**838.5**
Speedup	*1.46×*	*1.33×*	–	*1.22×*	*1.30×*	*1.61×*	*1.25×*	*1.50×*	*1.51×*

Table 1: Kimi-K2.5 (1T): batch completion time (s), input 16 requests. SGLang: default setting; SGLang-Opt: tuned for higher throughput. Speedup: BatchGen over SGLang-Opt. BatchGen is the only system serving 255K on one node.

End to end results - Flexibility

GPU Memory Constrained

Device: A5000(24GB HBM) + 1TB host memory
Workload:

- GSM8K(8.5k), input 512, output 256.
- ChatBot(36k), input 256, output 512.

→Models	Mixtral-8×7B		Mixtral-8×22B		DeepSeek-R1-671B	
→Datasets	GSM8K	ChatBot	GSM8K	ChatBot	GSM8K	ChatBot
vLLM	20.4h	N/A	375.0h	N/A	N/A	N/A
DeepSpeed	26.0h	187.4h	116.0h	950.4h	N/A	N/A
FlexGen†	18.8h	156.0h	83.5h	789.0h	N/A	N/A
MoE-Lightning†	7.3h	58.5h	32.5h	295.0h	N/A	N/A
BatchGen‡	**1.7h**	**10.0h**	**5.1h**	**30.8h**	**41.3h**	**328.5h**

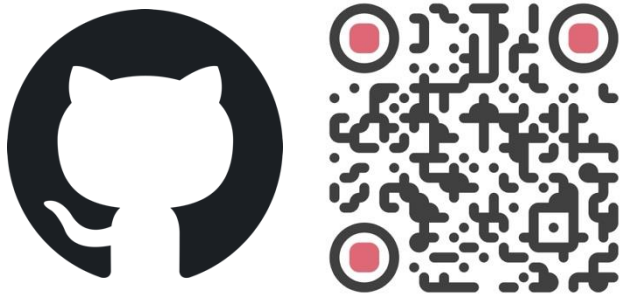
End to end results - Flexibility

Operational Cost

P:D Ratio	1:7	2:6	3:5	4:4	5:3	6:2	7:1
PD Disagg. (min)	137.9	67.4	49.0	38.6	40.1	55.6	112.0
BatchGen (min)				**17.5**			
Speedup	*7.9×*	*3.9×*	*2.8×*	*2.2×*	*2.3×*	*3.2×*	*6.4×*

Table 6: Comparison with PD disaggregation on 128×H20 GPUs (10K requests, 8K-2K workload). Each ratio unit is 16 GPUs. BatchGen requires no ratio tuning.

Summary



- Batch Inference is an important workload.
- Two main issues: underutilization & task/HW diversity
- We propose sequence coroutine abstraction and BatchGen Runtime
  - Analogy: Apache (process per connection) -> NGINX / SEDA (event-driven)

- Currently 1.2x-1.6x gain in long-context prefill, up to 9.6x for memory limited HW.
- Less extra cost in cluster configuration.