MACIEJ BESTA

28.05.2025

Graphs & LLMs: Synergy

Project Overview & Management

Incoming person: Lorenzo Paleari
(MSc in Computer Science @ ETH Zürich)

NeurIPS submission (May 22nd)

Parallel & Distributed RLMs

	Framework/Strategy	Data Parallelism	ZeRO-1	ZeRO-2	ZeRO-3	Tensor Parallelism	Sequence Parallelism	Context Parallelism	Pipeline Parallelism	Expert Parallelism
1	Transformer Reinforcement Learning (TLR)	🟢	🔴	🔴	🟢	🔴	🔴	🔴	🔴	🔴
2	ColossalChat (CAIChat)	🟢	🔴	🟢	🟢	🔴	🔴	🔴	🔴	🔴
3	DeepSpeed-Chat (DSChat)	🟢	🔴	🔴	🟢	🟢	🔴	🔴	🔴	🔴
4	OpenRLHF	🟢	🔴	🔴	🟢	🟢	🔴	🔴	🔴	🔴
5	HybridFlow (VeRL)	🟢	🔴	🔴	🟢	🟢	🔴	🔴	🟢	🔴
6	NeMo-Aligner	🟢	🔴	🔴	🔴	🟢	🔴	🔴	🟢	🔴
7	ReaLHF	🟢	🔴	🔴	🟢	🟢	🔴	🔴	🟢	🔴
8	RLHFuse	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🟢	🔴
9	FlexRLHF	🟢	🔴	🔴	🟢	🔴	🔴	🔴	🟢*	🔴
10	AsynchronousRLHF	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🔴	🔴
11	Pipe-RLHF (Computation Mode-Aware)	🟢	🔴	🔴	🔴	🔴	🔴	🔴	🟢	🔴

Parallel & Distributed RLMs

RLHF main components: policy generation, reward scoring, model updates

Traditional RLHF runs all these in sequence on the same devices, which can leave GPUs underutilized and elongate training time. Newer approaches seek to decouple and parallelize these stages across different devices.

We can categorise these strategies into a few **macro-level parallelism** techniques specific to RLHF on large language models

Parallel & Distributed RLMs

Disaggregated Model Placement: Distributing the RLHF components (policy model, reward model, value model, etc.) onto separate devices or nodes, rather than co-locating them on the same hardware.

Pros: Allows specialization of hardware (e.g., a GPU with more memory for the large policy model, and another for the reward model). Lays the groundwork for concurrent execution (since models are already on separate hardware).

Cons: Communication overhead is introduced when passing data (generated text, reward scores) between machines. Scheduling the stages in a distributed setting becomes more complex (requires coordination to hand off data).

Parallel & Distributed RLMs

Off-Policy RLHF: Decoupling the data generation from the training updates entirely, running them in parallel and off-policy. The policy model (or a copy of it) generates experience continuously, while the main model is being updated on older experience

Pros: It can significantly shorten wall-clock training time (e.g. Mila's work matched final performance with 40% less time). It also decouples the system design – generation and training code can run as separate processes or even on different HW (e.g. TPU vs GPU).

Cons: Off-policy training complexity – the learner is not always training on on-distribution data from its current policy, which can affect stability. There is a risk that the policy diverges if the lag becomes too large or if the reward model and policy get out of sync. Requires careful algorithmic choices

Parallel & Distributed RLMs

Stage Fusion: Running successive stages of the RLHF loop (most importantly generation → reward scoring) in overlapping fashion. As soon as one part of the output from a stage is ready, it is passed to the next stage without waiting for the entire batch/episode to complete.

Pros: Improves device utilization and cuts idle time.

Cons: Requires splitting work into chunks that can be processed independently and out-of-order. The logic to partition and merge results becomes complex (e.g. ensuring the correct reward is assigned to the correct sample when results return asynchronously)

Parallel & Distributed RLMs

	Framework / Strategy	Disaggregated Model Placement	Off-Policy RLHF	Stage Fusion
1	Transformer Reinforcement Learning (TRL)	🔴	🔴	🔴
2	ColossalChat	🔴	🔴	🔴
3	DeepSpeed-Chat	🔴	🔴	🔴
4	OpenRLHF	🟢	🔴	🔴
5	HybridFlow (VeRL)	🔴	🔴	🔴
6	NeMo-Aligner	🔴	🔴	🔴
7	ReaLHF	🔴	🔴	🔴
8	RLHFuse	🟢	🔴	🟢
9	FlexRLHF	🟢	🟢	🔴
10	Asynchronous RLHF	🟢	🟢	🔴
11	Pipe-RLHF	🟢	🔴	🔴

Parallel & Distributed RLMs: Next Steps

More principled understanding of parallelization techniques in this domain (clear naming and architectural template overview needed)

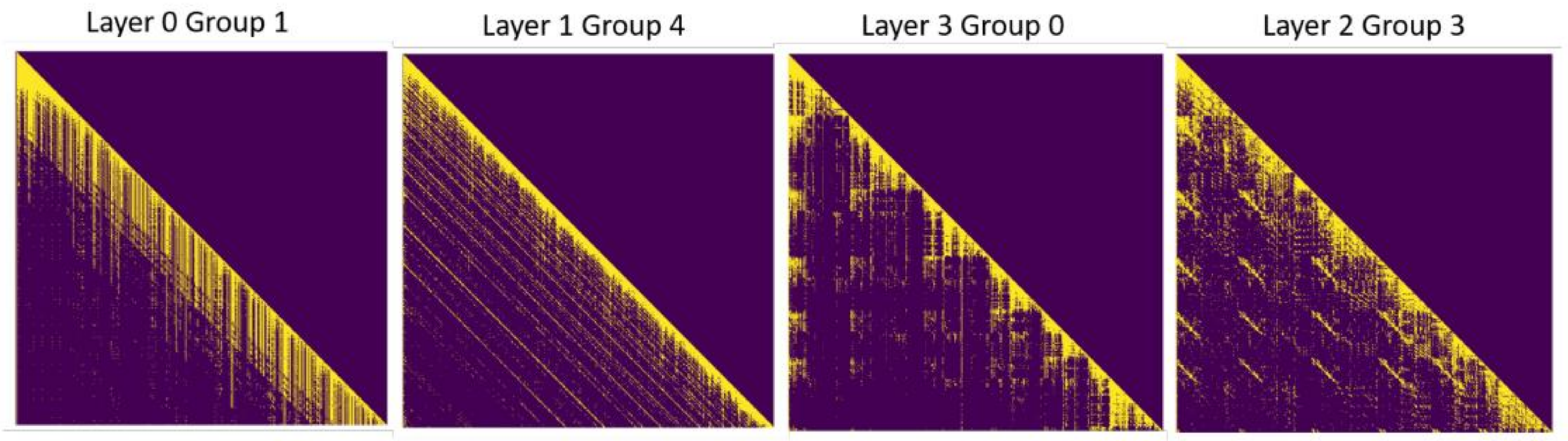
Performance modeling

Investigation into other forms of parallelism

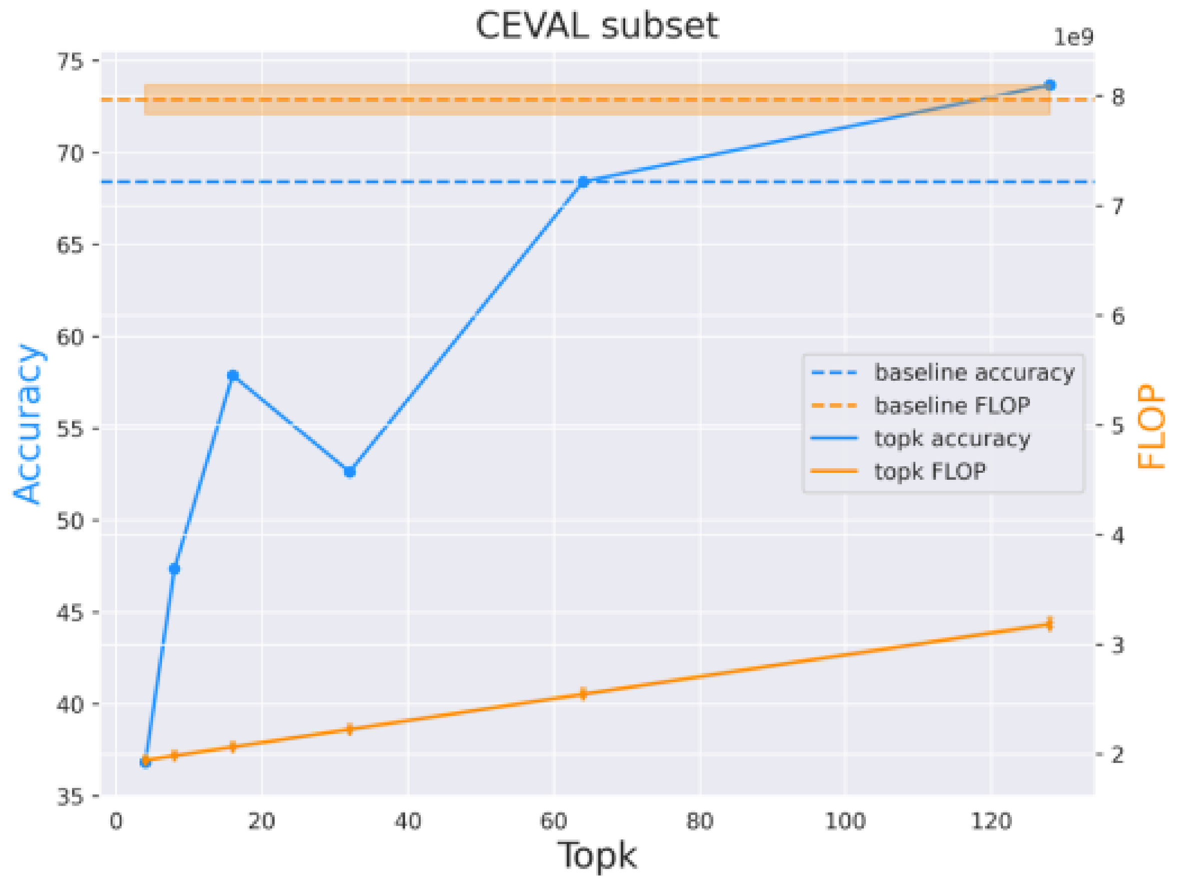
Chunk Sink

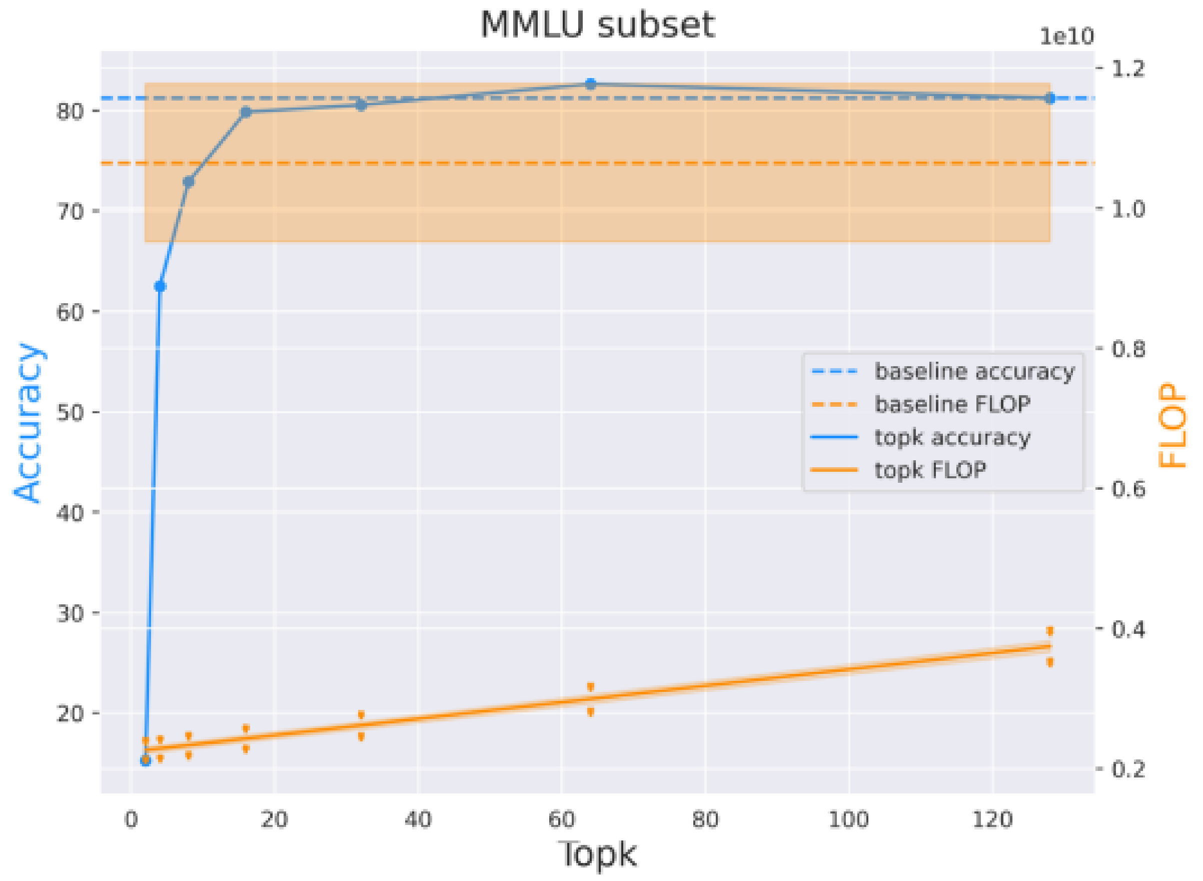
Insight: Attention naturally segments input into **semantic chunks**

Idea: caches only the first token of each semantically detected chunk, reducing KV cache size while preserving key contextual information.



Chunk Sink


CEVAL subset
Accuracy
FLOP
Topk
1e9
baseline accuracy
baseline FLOP
topk accuracy
topk FLOP
75
70
65
60
55
50
45
40
35
8
7
6
5
4
3
2
0
20
40
60
80
100
120


MMLU subset
Accuracy
FLOP
Topk
1e10
baseline accuracy
baseline FLOP
topk accuracy
topk FLOP
80
70
60
50
40
30
20
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100
120