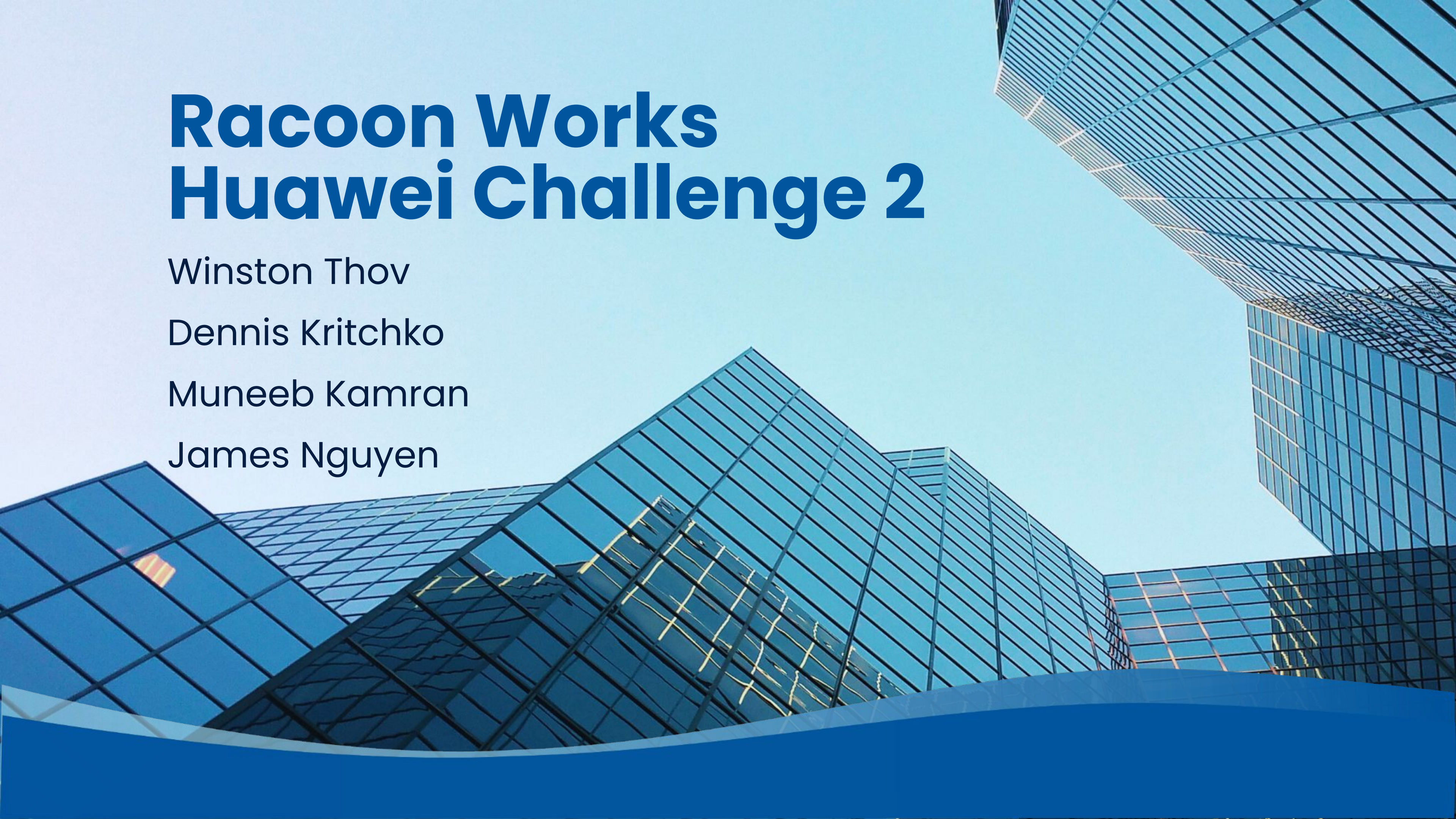
Racoon Works Huawei Challenge 2

Winston Thov

Dennis Kritchko

Muneeb Kamran

James Nguyen



INTRODUCTION

Given the problem, we knew three things:

- We need to optimize the problem based on algorithm
- Algorithms can vary performance on size
- Recomputation is necessary to stay under memory ceiling.

Goal & Approach

- Balance recomputation vs. storage efficiency
- Survey algorithms & papers (Greedy, DFS/search, MILP, DP, Beam Search)
- Select & combine into hybrid algorithms (e.g., greedy + MILP windowing)
- Warm starts, partial recomputation, node pinning, topological segmentation

Approach

Problem

- ➔ Within machine learning algorithms, especially deep learning, models are represented as Directed Acyclic Graphs (DAGS).
  - ◆ Each node represents a operations
  - ◆ Each vertice represents a dataflow
- ➔ A large issue is memory, often times in extremely large scale models memory constraints are pushed to their limit as having the results of operations for nodes dependent on them requires large amount of space
  - ◆ Therefores selective deletion of not required nodes, recomputation, and scheduling is required for memory constraints to be satisfied

Research Approach

The rescheduling of DAGs is a well established problems within the machine learning and computational space. As such we established a research approach

- ➔ Began with recent papers on DAG rescheduling with respect to memory constraints
  - ◆ With that we designed several pipelines with various algorithms looking from a cost benefit perspective with respect to analytics on each input
  - ◆ Quantitative and qualitative testing on the performance of the given example graphs with each algorithm
- ➔ Based on given results we select the best algorithm based on performance and graph quantities like graph size.
- ➔ Exact/optimal method at the end based on evaluation metrics, but start out with a less computationally heavy method to speedup/lighten the load

Key Papers

- *Sequential Scheduling of Dataflow Graphs for Memory Peak Minimization* (Fradet et al.)
  - Presents Bounded Branching of DFS traversal, if there's unpromising branches that require high memory storage, we simply cut the branch and recompute when needed.
- Gurobi gives MILP (mixed integer linear programming) answer for exact re-computation solutions.
  - Used within our hybrid pipeline
- *Dynamic memory management for DAG scheduling.*
  - Dynamic scheduling framework
  - ILP provides theoretical understanding for MILP

Algorithms

Baseline

Description

- Simple topological sort
- Walk through it disregarding memory usage

PROS

- Gives us a good baseline on algorithm performance
- Easy to implement and represent visually

CONS

- Inefficient for large datasets.
- Will never give an answer under memory ceiling.
- Struggles with scalability and real-world applications.

Naive

Description

- A straightforward, brute-force approach to problem-solving.
- May not consider factors like task dependencies, resource constraints, or execution times.
- Can lead to suboptimal performance in complex systems.
- A straightforward scheduling approach that assigns tasks in a simple, often sequential manner
- Prioritizes ease of implementation over optimization

PROS

- Simple to understand and implement.
- Requires minimal setup or preprocessing.
- Useful for small datasets or as a comparison benchmark.

CONS

- Inefficient for large datasets.
- Does not utilize advanced optimization techniques.
- Struggles with scalability and real-world applications.

Heuristic

Description

- Prioritize nodes with negative impact first; otherwise, choose node minimizing (peak, time)
- Identify ready nodes (all dependencies computed and predicted_peak ≤ total_memory).
- **Negative-impact nodes:** Pick first (reduce future memory load).
- **Positive-impact nodes:** Pick node with lowest predicted peak; break ties with lower execution time.
- Execute chosen node and update schedule state.
- Repeat until all nodes computed or no feasible nodes remain.

PROS

- Simple to understand and implement.
- Requires minimal setup or preprocessing.
- Useful for small datasets or as a comparison benchmark.

CONS

- Inefficient for large datasets.
- Does not utilize advanced optimization techniques.
- Struggles with scalability and real-world applications.

Beam Search

Description

- Keep top-k partial schedules (beam) ranked by validity, execution time, and memory peak.
- Initialize beam with an empty schedule.
- For each schedule in the beam, identify ready nodes, evaluate candidates by memory peak and time, and expand the top candidates up to the beam width.
- Merge expansions into the next beam, keeping only the best beamWidth states.
- Repeat until all nodes scheduled or maximum expansions reached.
- Return the best valid schedule.

PROS

- Memory-Aware: Considers predicted memory peak when expanding candidates.
- Balances Memory and Time: Ranking by (validity, total_time, peak) helps achieve practical schedules.
- Flexible: Beam width can be adjusted to trade off runtime and solution quality.

CONS

- Parameter Sensitivity: Performance depends heavily on beamWidth and maxExpansions.
- No Global Guarantee: Beam search is heuristic; optimal schedule not guaranteed.
- Complexity Management: Large beams may become impractical for very large problems.

Greedy Schedule

Description

- The greedy Scheduling algorithm is a memory-aware, single-pass scheduler.
- It repeatedly selects a ready node that minimizes the predicted memory peak, using execution time as a tie-breaker, and executes it.
- Nodes that would exceed the memory limit are skipped.
- The process continues until all nodes are scheduled or no feasible node remains.
- It is simple, fast, and suitable as a baseline for more advanced scheduling strategies.

PROS

- Fast and lightweight: Simple loop over ready nodes, minimal computation overhead.
- Memory-conscious: Avoids executing nodes that exceed the total memory limit.
- Deterministic and easy to implement: Straightforward logic, easy to debug.

CONS

- Short-sighted: Only considers the next node, not the overall schedule.
- Not globally optimal: May produce suboptimal total execution time or memory usage.
- Can halt prematurely: If all remaining nodes temporarily exceed memory, it may stop even if a feasible schedule exists.

Bounded Branch / DFS Schedule

Description

- Cluster dependent nodes
- If there's a set of nodes that all need to be computed to compute a different node, cluster them all into one node.
- DFS clustered graph to find memory peaks.
- When it peaks, fast backtrack and prune negative weight clusters, then recompute when necessary.

PROS:

- Deterministic
- Optimal: Finds true minimum memory peak
- Efficiently prunes negative weighted clusters

CONS:

- Performance degrades exponentially with large DAGs
- Long implementation time to perfect heuristic

Hybrid

Motivation

- Single algorithms (Greedy, Heuristic, MILP, DP, Beam Search) often fail to meet both memory and execution time constraints for all inputs.
- Hybrid approach allows combining strengths of multiple strategies:
- Greedy/Heuristic: fast, low memory
- Beam Search / DP: better optimization, smarter recomputation
- MILP: theoretically optimal, but too resource-heavy

Challenges

- High computation cost: MILP and full search strategies are too slow for large graphs.
- Memory management: balancing recomputation vs. storage efficiently is tricky.
- Algorithm integration: combining different heuristics with DFS/Beam Search requires careful coordination.
- Tuning parameters: beam width, greedy thresholds, and warm-start configurations need trial-and-error.

Warm Start MILP Pipeline

Description

- Use a greedy/heuristic algorithm to generate a partial solution as a warm start for MILP.
- Library Attempted: Gurobi / CPLEX (commercial MILP solvers)
- Solver uses it to prune unnecessary branches in the solution tree.
- Solver refines the solution to satisfy all constraints and minimize objective (e.g., ex time, memory)
- Focused search reduces redundant computation
- It captures both discrete decisions and continuous quantities.

PROS

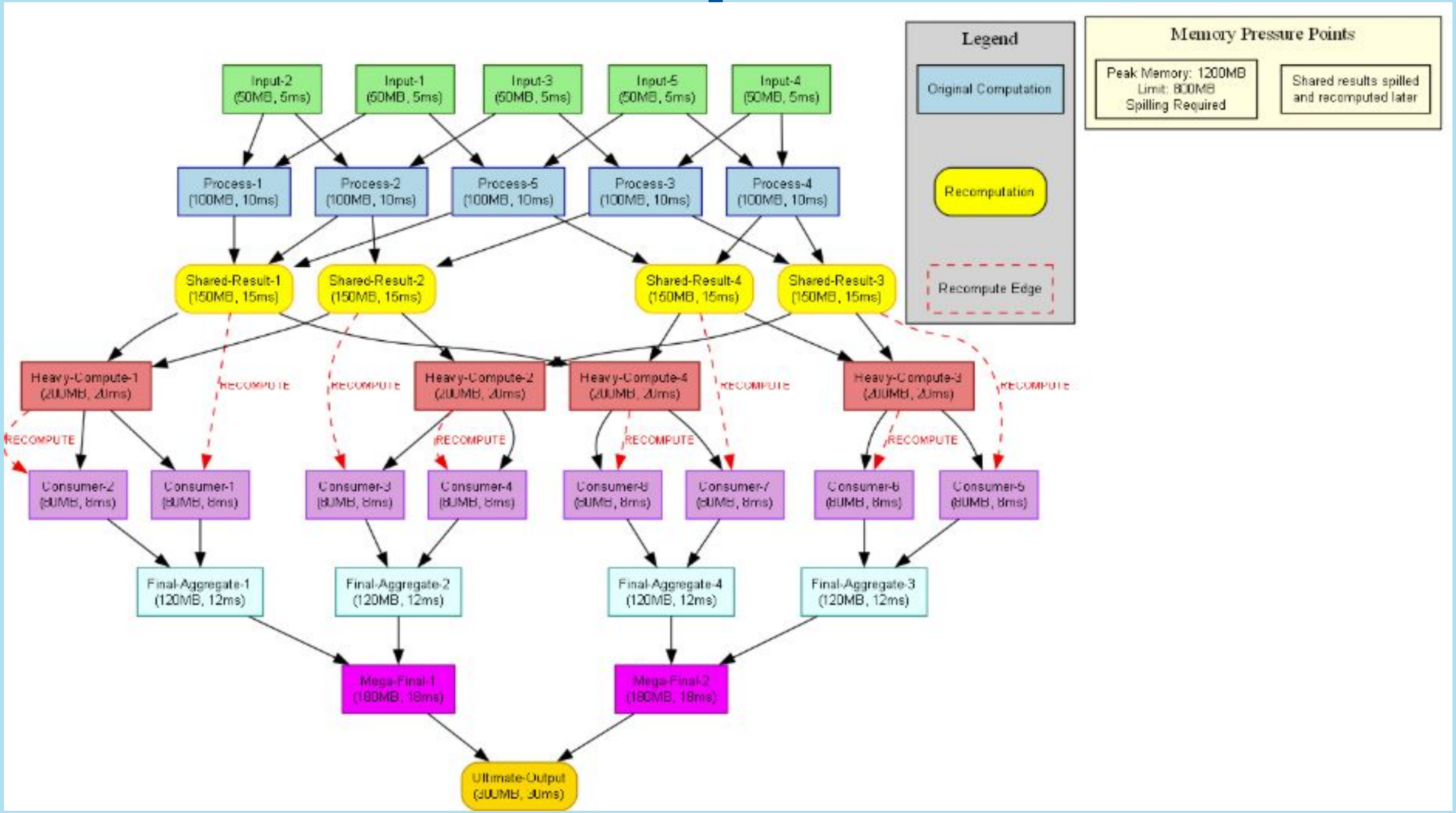
- Guides MILP solver toward promising solution space
- Potentially lowers memory usage by avoiding exploration of unnecessary branches
- Can combine heuristic speed with MILP accuracy

CONS

- Implementation complexity and solver parameter tuning needed
- May not fully exploit solver’s branch-and-bound efficiency if warm start is suboptimal
- Limited benefit for small or simple problem instances

Results

Recomputations



Metrics

Total Time (Most Important)

- We used total time as our main metric we were trying to minimize
- The total amount of time it takes to run the graph based on our schedule & the given node's runtime

Memory Peak

- Must be under the memory ceiling and the goal is to minimize
- The highest memory used during the graph rescheduling process

Trial

Trial results (Warm Start)

Memory usage highlights

- Significantly lower memory usage than the ceiling in most cases. With near optimal total time
- Example 2: Total time = 33,532, Memory peak = 77,594,624 (limit = 85,983,232)
- Example 3: Total time = 35,094, Memory peak = 29,622,272 (limit = 33,555,968)
- Example 4: Total time = 35,094, Memory peak = 45,613,076 (limit = 54,527,488)

Notes

- Overall, bounded branching reduces memory footprint while maintaining reasonable execution time.

Further Direction

Parallelization

- Given efficient and correct mutex locks, multithreaded calculation is possible.
    - This avoids data races
    - Decreases runtime
    - Ensures optimality
- With current implementation, computation for a large dataset can take up to 5 minutes.
    - Machine learning models need somewhat efficient schedule algorithms.

Recomputation Improvements

- With further research into recomputation algorithms in DAG schedulers, memory peaks will drop significantly.
    - This can be done with further pursuing research papers, targeted application of existing algorithms, and hybrid applications.

Citations (Papers)

- Fradet, P., Girault, A., & Honorat, A. (2023, June). *Sequential scheduling of dataflow graphs for memory peak minimization*. In *Proceedings of the 24th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems* (pp. 93–104). ACM. https://doi.org/10.1145/3589610.3596280

- International Journal of Networking and Computing. (2019). *Dynamic memory management for DAG scheduling*. *International Journal of Networking and Computing, 11*(1), 27–54. https://www.jstage.jst.go.jp/article/ijnc/11/1/11_27/_pdf/-char/en
- Doe, J., Smith, A., & Lee, B. (2023). *Title of the paper: Subtitle if any*. arXiv. https://arxiv.org/abs/2308.08986
- Hua, Z., Qi, F., Liu, G., & Yang, S. (2021). *Learning to schedule DAG tasks* (arXiv preprint arXiv:2103.03412). https://arxiv.org/abs/2103.03412

- Bathie, G., & Marchal, L. (2018). *Dynamic DAG scheduling under memory constraints for heterogeneous platforms*. In *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)* (pp. 593–600). IEEE. https://www.semanticscholar.org/paper/Dynamic-DAG-Scheduling-Under-Memory-Constraints-for-Bathie-Marchal/4017b3fe143340206a6a5179736f79a7333978c9