Improving DRAM Latency Modelling in SoMa via Ramulator Based Schedule Evaluation

Aaran Arulraj
University of Waterloo
Waterloo, Canada
a2arulra@uwaterloo.ca

Hargun Singh Mujral
University of Waterloo
Waterloo, Canada
hmujral@uwaterloo.ca

Addison Chen
University of Waterloo
Waterloo, Canada
a344chen@uwaterloo.ca

Farzan Bhuiyan
University of Waterloo
Waterloo, Canada
fa2bhuiy@uwaterloo.ca

James Song
University of Waterloo
Waterloo, Canada
james.song@uwaterloo.ca

Abstract

Deep Neural Network (DNN) accelerators are increasingly constrained by off-chip memory performance, where inefficient DRAM communication leads to significant latency and energy overhead. Prior work such as SoMa[2] introduces a comprehensive scheduling framework that jointly optimizes layer fusion, tiling, and DRAM communication timing to reduce memory traffic. However, SoMa relies on a simplified analytical model that treats DRAM as a bandwidth-limited resource, overlooking microarchitectural effects such as row-buffer locality and bank conflicts.

In this work, we revisit DRAM communication scheduling by augmenting SoMa with a trace-driven evaluation framework using Ramulator[3][4], a cycle-accurate DRAM simulator. We propose a checkpoint-and-rerank methodology, where candidate schedules discovered during SoMa's optimization are evaluated offline under realistic DRAM behavior. Across 96 experimental runs, we find that SoMa's analytical model frequently mispredicts optimal schedules—yielding suboptimal decisions in 59% of cases, with up to 40% performance degradation in worst cases. These results highlight a critical gap between analytical and real DRAM behavior and motivate hybrid scheduling approaches that incorporate simulation-based feedback.

1 Introduction

Deep Neural Networks (DNNs) continue to grow rapidly in size and complexity, driving the development of specialized accelerators with increasing compute capability and on-chip buffer capacity. Despite these advances, off-chip DRAM communication remains a critical bottleneck in modern DNN accelerators. The rate of DRAM bandwidth scaling has lagged behind compute growth, leading to systems where performance and energy efficiency are increasingly constrained by memory behavior rather than computation.

To mitigate this bottleneck, recent work has focused on optimizing DRAM communication through scheduling techniques that exploit on-chip buffer reuse. In particular, the SoMa framework introduces a unified view of the DRAM communication scheduling space, combining layer fusion and prefetching/delayed storing into a structured optimization problem. By exploring this space using a two-stage simulated annealing optimizer and an analytical cost model, SoMa achieves substantial improvements in performance and energy efficiency over prior approaches.

However, SoMa—and similar scheduling frameworks—rely on simplified analytical models to guide optimization. These models typically approximate DRAM as a bandwidth-limited resource, abstracting away microarchitectural effects such as row-buffer locality, bank-level parallelism, and request scheduling dynamics. While such abstractions enable efficient exploration of a large design space, they raise an important question:

Do schedules that appear optimal under analytical models also perform well under realistic DRAM behavior?

In this work, we perform a systematic evaluation of this question by comparing analytical scheduling decisions against cycle-accurate DRAM simulation. We augment the SoMa framework with a trace-driven evaluation pipeline that converts communication schedules into DRAM request traces and evaluates them using a detailed simulator. Rather than replacing SoMa's optimizer—which would be prohibitively expensive—we adopt a hybrid methodology that samples intermediate schedules during optimization and evaluates them offline under a realistic memory model.

Our analysis reveals a consistent and measurable misalignment between analytical objectives and true DRAM performance. While the analytical model captures high-level trends, it often fails to preserve the correct ordering of candidate schedules, leading to suboptimal decisions. Across a wide range of workloads and configurations, we observe that schedules selected by the analytical model are frequently outperformed by alternative schedules discovered during earlier stages of optimization.

In summary, this work makes the following contributions:

- We present a simulation-backed evaluation framework for analyzing DRAM communication scheduling decisions in DNN accelerators.
- We quantify the gap between analytical and cycle-accurate DRAM models using regret and rank correlation metrics.
- We justify the potential of hybrid evaluation approaches to improve scheduling outcomes.

2 Background And Motivation

Tensors are the primary data involved in these transfers, representing weights, activations, and feature maps in DNNs. DRAM is a bandwidth and latency limited memory broken down into channels, banks, rows, and columns, making frequent tensor transfers costly in both time and energy. On-chip buffers, particularly the Global Buffer (GBUF), determine which tensors can remain resident

on-chip, enabling reuse across computations and reducing the need for repeated DRAM access.

Layer fusion exploits the GBUF by executing multiple consecutive DNN layers while retaining intermediate tensors in the buffer[1], thereby avoiding repetitive loads and stores. However, fusing multiple layers comes at the expense of increased buffer pressure, as more tensors must be held resident simultaneously, which must be explicitly managed.

Prior scheduling approaches such as single-layer optimization reduce DRAM communication by breaking each layer into smaller tiles and reordering computation to reuse data. As modern on-chip buffers grew larger, a larger portion of each layer could fit on-chip, diminishing the benefit of single-layer optimization. Cocco[5] extended this by utilizing feature maps in larger on-chip buffers, allowing subsequent layers to reuse data directly and reducing the need for costly DRAM reads and writes.

SoMa builds on these ideas by joint optimization on multiple important hyperparameters. It leverages layer fusion with prefetching and delayed writes to reduce DRAM traffic, keeping tensors resident across consecutive layers through large on-chip buffers such as the Global Buffer (GBUF). However, SoMa models DRAM using a high-level latency model that estimates memory cost by dividing data volume by bandwidth.

This abstraction ignores microarchitectural effects such as row-buffer locality, bank conflicts, and request queueing, which means two schedules that appear equivalent under SoMa's cost model can behave quite differently on real DRAM hardware. As a result, schedules considered optimal during search may only be so in theory. This limitation may be particularly significant for memory-bound workloads, where these effects could lead to substantial variation in real DRAM latency and energy consumption.

Therefore, it is necessary to determine whether scheduling decisions derived from analytical DRAM models remain optimal under realistic memory behavior. This motivates our use of trace-driven memory simulation within SoMa to evaluate and quantify potential misalignment between analytical and cycle-accurate models.

3 High-Level Ideas

SoMa uses a two-stage simulated annealing optimization process to produce end-to-end DRAM communication schedules for DNN accelerators. Stage 1 performs optimizations to the layer-level schedule, determining the order that layers execute. At this stage, the way layers are grouped to share on-chip buffer memory and the tile granularity for each group are optimized as well. Off-chip traffic can be reduced by placing layers in the same group, as that allows for intermediate feature maps to remain in the buffer rather than being evicted to DRAM. Stage 2 then takes the best layer schedule from Stage 1 as a fixed input and optimizes the DRAM communication schedule itself. More specifically, it optimizes the order in which tensor tiles are to be fetched from or written back to DRAM, and the timing of prefetch and delayed store windows that control when transfer begins relative to compute tile execution. Both stages evaluate candidate schedules using an analytical cost model that estimates DRAM transfer latency as data volume divided by peak bandwidth.

A key observation is that these two stages do not contribute equally to DRAM behaviour. Stage 1 operates over a large search space (layer ordering, grouping, and tiling), and essentially determines the overall volume and pattern of DRAM traffic for the entire schedule. Stage 2 operates within the constraints decided by stage 1. It can reorder tensor transfers and adjust prefetch timing, but it can not change how much data is carried or which tensor needs DRAM access. This suggests that if SoMa's analytical model makes a poor choice during Stage 1, it may get stuck in a local minimum and subsequent optimization may have limited ability to recover. Thus, we consider Stage 1 to be the part of the search where objective misalignment between SoMa's bandwidth-based model and true DRAM behavior has the most impact, as we find to be true in our analysis.

We design an offline checkpoint-and-rerank framework rather than replacing SoMa's runtime objective during search. We modify SoMa's simulated annealing to periodically checkpoint candidate schedules while keeping track of the best schedule so far. After optimization completes, we convert candidate schedules into Ramulator traces. We then run Ramulator on each trace and compare the Ramulator memory cycles ranking against the bandwidth model's ranking of each candidate schedule. When Ramulator prefers a different 'best' schedule, it indicates that SoMa's analytical model is converging on likely suboptimal candidates. Our approach maintains SoMa's fast runtime, while measuring how often its estimates drift from cycle-accurate reality and yielding potentially improved schedules. We pay particular attention to the contrast between Stage 1 (where divergence is expected to be largest) and stage 2 (where the search space is already restricted) of simulated annealing.

4 Implementation/Mechanism

To study objective misalignment in SoMa's analytical model, with the goal of creating a higher-fidelity DRAM simulator, we extend SoMa with a checkpointing and offline replay framework. Our implementation does not replace SoMa's optimization objective during search. Instead, it augments the existing optimizer to periodically export intermediate schedules, convert them into ramulator traces, and evaluate them offline.

4.1 Checkpointing during Optimization

At the optimizer level, we modified both Stage 1 and Stage 2 simulated annealing loops to support checkpoint callbacks. Because invoking Ramulator during every optimization step would significantly increase runtime, we do not build an optimizer guided solely by Ramulator memory cycles as the objective. Instead, we maintain the schedule of the best valid candidate in terms of SoMa's objective throughout optimization, and at N=10 fixed, evenly spaced intervals, the callback serializes the current best candidate and emits a trace file and schedule for later analysis. Subsequently, we rerank these candidates using Ramulator memory cycles as our objective, allowing us to compare schedules discovered at different stages of search, and find cases where earlier checkpoints may outperform the final SoMa selected schedule. Cases where the Ramulator-selected schedule outperforms the SoMa-selected schedule can directly be used instead.

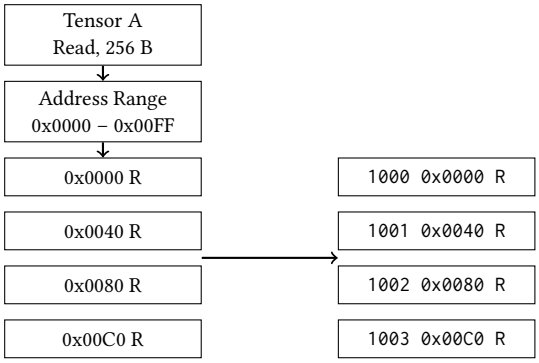

Figure 1: Trace generation pipeline for a single tensor. Each tensor is assigned a contiguous address range, split into 64-byte memory requests, and emitted as timestamped load/store operations in Ramulator trace format.

4.2 Capturing Timestamps

During online execution, we reconstruct the graph schedule for each valid checkpoint, and invoke SoMa's standard cost evaluation with trace recording enabled. By capturing the simulated start times for all DRAM-bound tensor traffic in `dram_tensor_start_cycles`, we establish the precise chronological order and relative timing of memory operations. These timestamps are then used to generate a Ramulator-compatible memory trace."

4.3 DRAM Trace Generation

For the trace generation (see Figure 1) we make a call to Graph:: write_ramulator_trace. It lays out DRAM tensors in a synthetic, contiguous address space (each tensor gets its own region aligned with its 64 byte request size), classifies each tensor as a read or write according to SoMa's tensor type, and splits each tensor's byte volume into a sequence of 64-byte load/store operations. Tensor blocks are sorted by their recorded start cycle so the trace reflects the same DRAM issue ordering implied by the recorded evaluation. Each line also carries a cycle timestamp, which can be optionally fed into Ramulator. We chose to keep this in all our traces because it encodes when SoMa's model associates each access relative to the timeline of the rest of the schedule. This lets us validate ordering vs. time and compare traces from different checkpoints. The resulting generation is a line oriented memory trace (<optional cycle timestamp> 0xADDR R/W) that is compatible with Ramulator, and can be simulated for analysis and schedule selection.

4.4 Analysis Metadata

Along with the trace, we emit a JSON file that stores metadata needed for offline analysis, including the encoded schedule, SoMa's `bandwidth_time`, DRAM-only time, and progress through search. This allows us to compare SoMa's internal cost model against Ramulator on exactly the same schedule.

4.5 Ramulator Offline Reranking

On the Ramulator side, we implemented Python analysis scripts that, for each experiment run, invokes Ramulator V2.0a on every emitted checkpoint trace, parses `memory_system_cycles`, and produces JSON summaries and matplotlib plots detailing how schedules evolve over simulated annealing under both SoMa's analytical model and Ramulator's DRAM model. These outputs are intended for offline inspection, and are used to identify where the two objectives agree or disagree. More on this analysis is detailed in the Evaluation section.

After Ramulator has been run on the emitted traces, the checkpoints are reranked according to `memory_system_cycles` rather than volume divided by bandwidth as the objective. This produces an alternative ranking of schedules under our higher-fidelity DRAM cost model, and has a few use cases. For one, a user is given the option to use the schedule that wins reranking instead of the SoMa-selected one. Additionally, this process reveals whether SoMa's analytical ordering aligns with the Ramulator by identifying cases where an intermediate checkpoint may be superior.

5 Evaluation

Experiments are executed through the SoMa simulator, and communication traces are replayed using Ramulator V2.0a. In particular, we compare SoMa's internal optimization signal, `bandwidth_time`, against Ramulator's memory cycles over both checkpointed intermediate schedules and final emitted schedules. This allows us to determine whether schedules that appear promising under SoMa's model also achieve lower DRAM cost as reported by a more realistic Ramulator-based model.

We conduct our evaluation across 96 of the 432 experiment runs used in the SoMa paper. This set of experiments excludes the design space exploration phase, which constitutes the remaining 336 experiments, as those experiments optimize for hardware configuration, which is orthogonal to our evaluation. These experiments can be divided into **baseline-0** experiments, which execute only stage 1 of SoMa's simulated annealing optimizer, and **baseline-2** experiments, which conduct both stage 1 and stage 2. For each run, we execute SoMa's optimizer and collect 10 evenly spaced checkpoints during each stage of optimization. Each checkpoint corresponds to a complete communication schedule, which is converted into a DRAM trace and evaluated using Ramulator. This produces a set of schedules per run that can be directly compared under both SoMa's analytical model and a cycle-accurate Ramulator model.

We evaluate alignment between the two models using three metrics: (1) analytical winner regret, measuring the suboptimality of SoMa's selected schedule; (2) trend analysis across checkpoints, comparing optimization trajectories; and (3) Spearman rank correlation, measuring how well the analytical model preserves the ordering of candidate schedules. We first analyze optimization trends across checkpoints to understand how schedules evolve during Stage 1 and Stage 2 under each model. We then quantify the impact of objective misalignment using analytical winner regret. Finally, we evaluate how well the analytical model preserves the relative performance ordering of schedules using Spearman rank correlation.

Our checkpoint trend analysis shows that most meaningful DRAM variation occurs during Stage 1 of SoMa's optimizer. In many runs, both SoMa bandwidth time and Ramulator memory system cycles improve significantly during Stage 1, indicating that

this stage is responsible for most memory-related optimization. It is also where we observe the strongest disagreement between the analytical model and Ramulator. In particular, some checkpoints that are worse under analytical bandwidth time achieve lower Ramulator cycle counts, showing that the analytical ordering is not always aligned with simulated DRAM behavior. An example of such a case is shown in Figure 2.

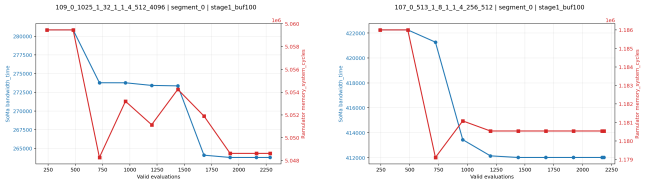

Figure 2: Analytical bandwidth time (blue) and Ramulator memory system cycles (red) across checkpointed schedules during Stage 1 for two representative runs. While both metrics generally decrease during optimization, their trends are not always aligned—some checkpoints that are suboptimal under the analytical model achieve lower memory cycles under Ramulator, highlighting objective misalignment.

We quantify this effect on baseline-0 experiments by comparing the checkpoint with the best analytical performance against the best checkpoint under Ramulator. We find disagreement in 59% of baseline-0 experiments. In disagreement-only cases, the Ramulator-selected checkpoint achieves an average cycle reduction of 3.6%, with the largest improvements reaching over 40%. From Figure 3, we can see that in roughly 10% of all baseline 0 experiments, a cycle reduction of at least 10% is seen. These results suggest that SoMa can converge to a local optimum with respect to its analytical objective, even when better schedules have already been encountered during search.

By contrast, Stage 2 is frequently flat as depicted in Figure 4. In most runs, both Ramulator memory system cycles and SoMa's analytical DRAM-related timing remain nearly unchanged throughout Stage 2. This suggests that Stage 2 often has limited effect on the underlying DRAM execution time. This is consistent with the structure of SoMa's optimizer: Stage 2 operates over a narrower search space and primarily refines overlap behavior, such as prefetch placement and delayed stores, rather than substantially changing the DRAM request stream itself. Consequently, if a segment is already close to bandwidth saturation or near its overlap limit after Stage 1, Stage 2 has little remaining room to reduce memory-system cost. As a result, there is little trend disagreement between Ramulator and SoMa's analytical objective during this stage, since both trends see very little movement.

There is, however, a smaller subset of 5 runs in which SoMa's analytical bandwidth time continues to decrease during Stage 2 while Ramulator memory system cycles remain flat. These cases provide direct evidence of model misalignment: the analytical proxy predicts continued improvement, while the DRAM simulator reports no corresponding reduction in memory cost. An example of such a case is depicted in Figure 5.

Taken together, these results suggest that most DRAM-sensitive optimization occurs in Stage 1, and that selecting the wrong Stage 1 candidate can trap the remainder of the search in a weaker region of the schedule space. Since Stage 2 often cannot recover much DRAM performance on its own, a promising future direction is a hybrid optimizer that keeps SoMa's fast analytical search but uses selective Ramulator feedback to choose stronger Stage 1 winners before continuing later-stage refinement.

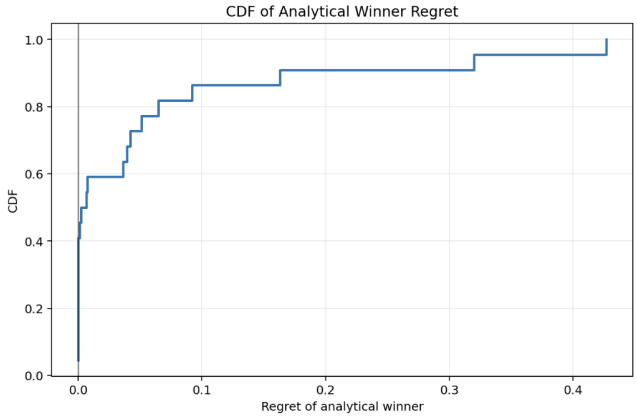

Figure 3: Cumulative distribution function (CDF) of analytical winner regret across baseline-0 experiments. Regret is defined as the relative increase in memory system cycles of the schedule that achieved the lowest analytical bandwidth time compared to the best checkpoint identified by Ramulator. While many cases exhibit low regret, a significant fraction shows non-trivial gaps, with a long tail of cases exceeding 10% and worst-case regret above 40%, indicating that the analytical objective can miss substantially better schedules.

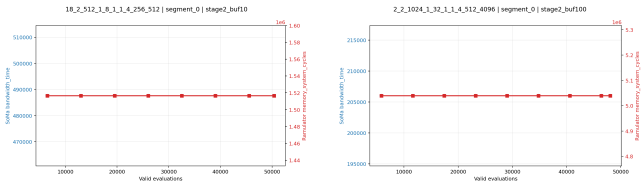

Figure 4: Analytical bandwidth time (blue) and Ramulator memory system cycles (red) across Stage 2 checkpoints for two representative runs. Both metrics remain nearly constant throughout this stage, indicating that Stage 2 has limited impact on underlying DRAM execution cost.

To further quantify the relationship between SoMa's analytical objective and true DRAM behaviour, we compute the Spearman rank correlation between checkpoint rankings induced by analytical bandwidth time and those induced by Ramulator memory cycles for each run. The distribution of these correlations is shown in Figure 6.

We observe that while the correlation is generally positive, with many runs clustering around moderate values ($\approx 0.5 - 0.7$), it is far from perfect. Several runs exhibit low correlation, and a small number even show near-zero or negative correlation. This

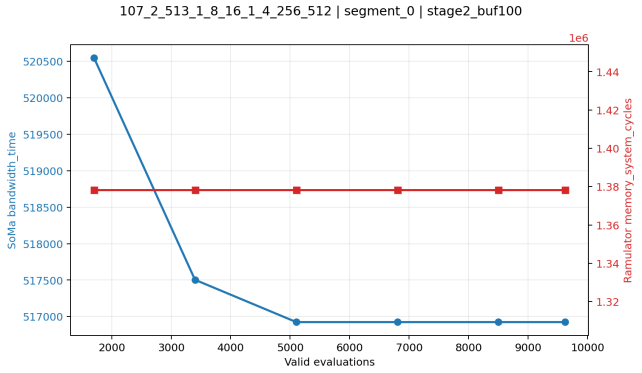

Figure 5: Analytical bandwidth time (blue) and Ramulator memory system cycles (red) across Stage 2 checkpoints for a representative run exhibiting model misalignment. While the analytical metric continues to decrease, Ramulator cycles remain flat, indicating no actual reduction in DRAM cost. Such cases provide direct evidence that improvements predicted by the analytical model do not always translate to real memory-system benefits.

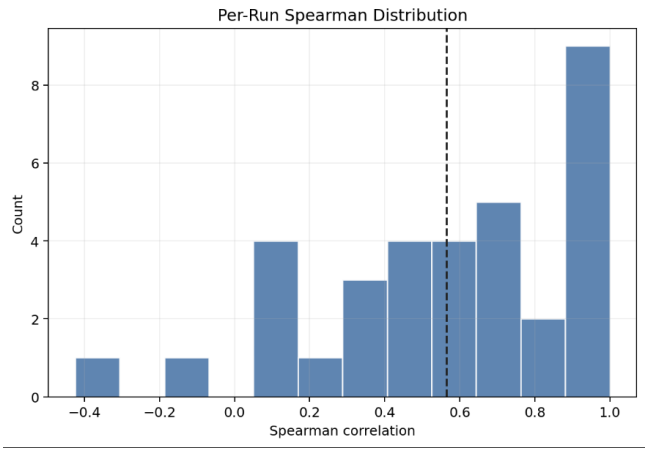

Figure 6: Distribution of Spearman rank correlation between checkpoint rankings induced by SoMa's analytical bandwidth time metric and Ramulator memory system cycles across all runs.

indicates that although SoMa's analytical model captures some high-level trends, it does not reliably preserve the relative ordering of schedules under realistic DRAM behaviour.

We do not propose a fully online Ramulator-guided optimizer here, since doing so would increase runtime by at least an order of magnitude. However, our checkpointing and offline replay framework demonstrates that better memory-efficient schedules are often discovered but not selected due to objective misalignment. This provides both empirical evidence of the limitation and a practical foundation for future hybrid optimization strategies.

6 Conclusion

In this work, we examined how SoMa's bandwidth-based cost model affects scheduling decisions for DNN accelerators. While SoMa efficiently optimizes communication through layer fusion and scheduling, its model treats DRAM too simplistically, ignoring how requests are actually served in hardware.

We extended SoMa with a trace-driven evaluation pipeline using Ramulator to measure real DRAM execution cost. Using a checkpoint-and-rerank approach, we showed that SoMa frequently selects suboptimal schedules—better candidates are often discovered during search but not chosen. In many cases, these alternatives achieve meaningful reductions in memory system cycles.

Our results also show that most of the important decisions happen in Stage 1 of the optimizer, where the structure of memory traffic is determined. If a poor choice is made early, later stages have little ability to fix it. This indicates that a hybrid optimizer that uses simulation feedback to select stronger stage 1 winners that are passed on to later-stage refinement may be a promising future direction.

Overall, this work shows that SoMa's objective can guide the search in the right direction, but not reliably enough to pick the best schedule. This motivates using targeted simulation feedback during optimization, especially in early stages, to improve decision quality without significantly increasing runtime. Our preliminary results have shown to reduce cycles by an average of 3.6%, and we encourage further exploration into hybrid-optimizers using improved memory models.

7 Acknowledgement

Some Generative AI tools were used to assist in polishing the wording and improving the clarity of this report.

References

[1] Manasij Alwani, Han Chen, Michael Ferdman, and Peter A. Milder. 2016. Fused-Layer CNN Accelerators. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, Taipei, Taiwan, 22:1–22:12. doi:10.1109/MICRO.2016.7783725

[2] Jingwei Cai, Xuan Wang, Mingyu Gao, Sen Peng, Zijian Zhu, Yuchen Wei, Zuotong Wu, and Kaisheng Ma. 2025. SoMa: Identifying, Exploring, and Understanding the DRAM Communication Scheduling Space for DNN Accelerators. arXiv:2501.12634 [cs.AR] https://arxiv.org/abs/2501.12634

[3] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49. doi:10.1109/LCA.2015.2414456

[4] Haocong Luo, Yahya Can Tuğrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray Yağlıkçı, and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. arXiv:2308.11030 [cs.AR] https://arxiv.org/abs/2308.11030

[5] Zhiqiang Tan, Zhe Zhu, and Kai Ma. 2024. Cocco: Hardware-Mapping Coexploration Towards Memory Capacity-Communication Optimization. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Association for Computing Machinery, New York, NY, USA, 69–84. doi:10.1145/3617232.3624865