# Multi-Agent Reinforcement Learning for Microprocessor Design Space Exploration

**Srivatsan Krishnan**[*]
Harvard University
&
Google Research
Brain Team

**Natasha Jaques**
Google Research
Brain Team

**Shayegan Omidshafiei**
Google Research
People + AI Research Team

**Dan Zhang**
Google Research
Brain Team

**Izzeddin Gur**
Google Research
Brain Team

**Vijay Janapa Reddi**
Harvard
University

**Aleksandra Faust**
Google Research
Brain Team

## Abstract

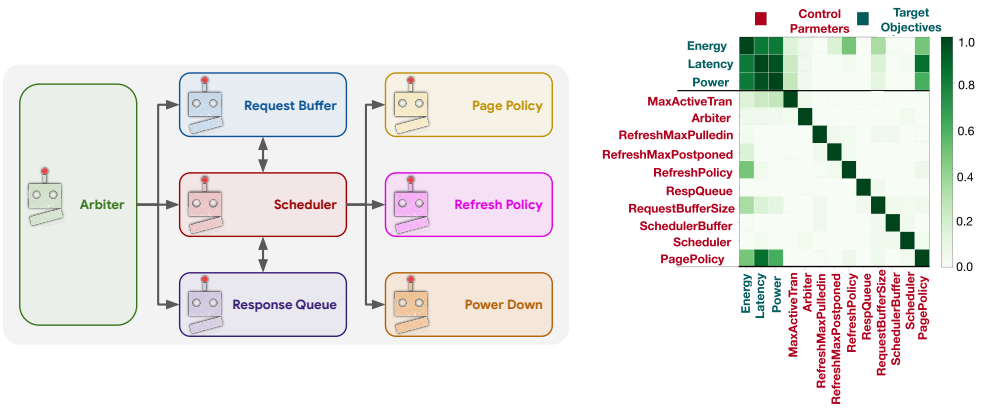
Microprocessor architects are increasingly resorting to domain-specific customization in the quest for high-performance and energy-efficiency. As the systems grow in complexity, fine-tuning architectural parameters across multiple sub-systems (e.g., datapath, memory blocks in different hierarchies, interconnects, compiler optimization, etc.) quickly results in a combinatorial explosion of design space. This makes domain-specific customization an extremely challenging task. Prior work explores using reinforcement learning (RL) and other optimization methods to automatically explore the large design space. However, these methods have traditionally relied on single-agent RL/ML formulations. It is unclear how scalable single-agent formulations are as we increase the complexity of the design space (e.g., full stack System-on-Chip design). Therefore, we propose an alternative formulation that leverages Multi-Agent RL (MARL) to tackle this problem. The key idea behind using MARL is an observation that parameters across different sub-systems are more or less independent, thus allowing a decentralized role assigned to each agent. We test this hypothesis by designing domain-specific DRAM memory controller for several workload traces. Our evaluation shows that the MARL formulation consistently outperforms single-agent RL baselines such as Proximal Policy Optimization and Soft Actor-Critic over different target objectives such as low power and latency. To this end, this work opens the pathway for new and promising research in MARL solutions for hardware architecture search.

## 1 Introduction

Computer systems are complex and interconnected systems, where various sub-components interact with each other to execute a program successfully. Due to the slowdown of Moore's law [Schaller, 1997] and end of Dennard scaling [Esmaeilzadeh et al., 2011], computer system architects are increasingly resorting to domain-specific customization [Dally et al., 2020] of various sub-components in the computer system (e.g., compute, cache, memory, interconnects) to achieve better performance under stringent energy efficiency targets. As the customization boundary extends to the whole computer system stack (i.e., datapath, memory, interconnects, compilers, and application), it also results in a combinatorial explosion in the number of possible custom designs. For instance, enumerating the full

[*]Work done, in part, via the Google Student Researcher program. Contact at srivatsan@seas.harvard.edu

36th Conference on Neural Information Processing Systems (NeurIPS 2022).




(a) Interconnected components and MARL formulation.

(b) Correlation matrix.

Figure 1: (a) DRAM memory controller as an example of an interconnected system. Also indicated is the proposed MARL formulation, involving each agent controlling a distinct system component. (b) $\phi_k$ correlation [Baak et al., 2018] between memory controller parameters and target objectives.

stack for a custom DNN accelerator increases the overall parameter space to $10^{2300}$ [Zhang et al., 2022].

**Related Work and Current Approach.** The popularity [Sutton and Barto, 2018] of reinforcement learning (RL) continues to rise, and computer architects have successfully applied RL for the parameter selection problem. For instance, RL-based parameter selection was applied for designing a custom scheduling policy in memory controllers [Ipek et al., 2008], data path for DNN accelerators [Yazdanbakhsh et al., 2022, Kao et al., 2020], Network-on-Chip (NoC) topology [Lin et al., 2020], and transistor sizing [Wang et al., 2020, Settaluri et al., 2020]. However, these RL formulations are single-agent based, where a single policy simultaneously determines all system parameter values. In these scenarios, the policy needs to learn the observation to action mapping and the relationship between various parameters in all components of the interconnected system. As the complexity of the computer system increases, it is unclear if single-agent formulations will scale accordingly.

**Our Contribution.** We introduce Multi-Agent RL (MARL) to tackle the problem of microprocessor hardware architecture search in a scalable and systematic manner. To the best of our knowledge, this is the first multi-agent formulation to tackle architecture design space exploration. To apply our MARL formulation, we use a DRAM memory controller as an example of an interconnected system shown in Figure 1a. A DRAM memory controller consists of `Request Buffer`, `Response Buffer`, `Arbiter`, `Scheduler`, and policy control managers for determining `Page Policy`, `Refresh Policy`, and `PowerDown Policy`. Each of these subsystems can have many parameters which eventually determine the latency and power consumption for the DRAM subsystem. Thus, the DRAM memory controller plays a key role in determining the performance of the memory subsystem.

**Case for Decentralized MARL.** The key idea behind using a decentralized MARL approach is that the parameters in a computer system are more or less independent (are not strongly correlated with each other). Intuitively, for high-performance and energy-efficient designs, we want each component to be independent since having more than one sub-components achieve the same purpose means having redundancy in the system. Though designing a redundant system is a valid design goal, our focus in this work is to design customized hardware architectures for high performance and efficiency.

To empirically validate that the control parameters are independent, we perform a random grid search over the DRAM memory controller parameters and their impact on overall latency, energy, and power. Using this data, we construct the correlation matrix as shown in Figure 1b between various parameters in a DRAM memory controller system. We observe that the control parameters have a very low correlation between them (darker shades of green mean more correlation) which makes it amenable to formulating the problem using a decentralized multi-agent approach. Note that this also motivates using decentralized multi-agent PPO [de Witt et al., 2020] for architecture DSE, rather than centralized training, decentralized execution multi-agent RL formulations such as MAPPO [Yu et al., 2021] or MADDPG [Lowe et al., 2017].

A key benefit of using MARL, rather than single-agent RL, is to divide and conquer large parameter design spaces. The MARL formulation provides the necessary constructs and infrastructure to assign fine-grained roles to many agents. Each agent implicitly communicates with other agents through shared rewards and observations to converge on a set of parameters such that the team meets the global design objective (e.g., low latency or low power).

In our MARL formulation for the DRAM memory controller, we assign each agent to control one specific memory controller parameter. For example, as shown in Figure 1a, the *Request Buffer Agent* controls the `Request Buffer` size. Similarly, the *Page Policy Agent* controls the `Page Policy` parameter. Since the parameter space is divided across each independent agent, the MARL formulation allows for factorizing the large design space across many agents.

**Domain-Specific Custom Memory Controller Design.** To validate the effectiveness of our MARL formulation, we design a custom memory controller for two different memory access patterns for lowering its latency and power.For estimating the cost in terms of latency, power, and energy for selecting specific memory controller parameters, we use DRAMSys [Steiner et al., 2020]. DRAMSys is a transaction-level simulator that takes memory traces as inputs and outputs the latency, power, and energy for a given memory controller parameters. We construct a DRAM-Gym environment to evaluate our MARL formulation and variants of single-agent RL algorithms such as Proximal Policy Optimization (PPO) [Schulman et al., 2017] and Soft Actor-Critic (SAC) [Haarnoja et al., 2018]. PPO and SAC are state-of-the-art RL algorithms that have shown promising results in several sequential decision-making tasks. We use the ACME [Hoffman et al., 2020] as the RL framework for the MARL and other single-agent baselines.

**Contributions.** We make the following contributions: (1) First demonstration of applying MARL for the microprocessor hardware architecture search problem, (2) We demonstrate our MARL formulation consistently outperforms single agent RL formulations in terms of designing a domain-specific memory controller for multiple memory traces and objective targets, (3) Our MARL formulations achieve a mean episodic reward that is $\approx 2\times$ to $60\times$ more than single-agent formulations.

## 2 Methodology

In this work, we propose a MARL formulation for DRAM memory controller parameter exploration. We compare the performance of MARL formulations with single agent formulations such as PPO [Schulman et al., 2017] and SAC [Haarnoja et al., 2018]. In particular, for the MARL formulation, we use multi-agent decentralized PPO since it has shown state-of-the-art performance in various sequential decision-making tasks [de Witt et al., 2020, Yu et al., 2021].

**Decentralized MARL.** We have decentralized agents taking simultaneous independent actions. For the domain-specific custom memory controller design, we have ten parameters (see Appendix 5.2), each controlled by a distinct agent. For instance, the size of the `Request Buffer` is controlled by Request Buffer Agent whereas `Page Policy` is controlled by Page Policy Agent, as shown in Figure 1a. Each agent receives the same observation and a shared reward signal. Please refer to Appendix 5.3 for more information.

**Single-Agent Baselines.** For single-agent, we have three different baselines, namely PPO [Schulman et al., 2017], SAC [Haarnoja et al., 2018], and what we define as Time Division Multiplexing (TDM). We use the ACME [Hoffman et al., 2020] framework for PPO and SAC, wherein a single agent takes ten actions simultaneously (see Appendix 5.2). Hyperparameters for these agents are tabulated in Appendix 5.3.

TDM is the most basic form of RL formulation [Sutton and Barto, 2018], where a single agent produces one action each time step. In the case of the DRAM memory controller, there are ten parameters (refer to Appendix 5.2); thus, the TDM agent will predict `Page Policy` parameter at timestep one, the `Request Buffer Size` parameter at timestep two, and so on. At the end of ten timesteps, all the parameters accumulated are sent to the DRAM-Gym environment. Since there is a single policy controlled by a single agent (whose role changes on each timestep), we denote this formulation Time Division Multiplexing. Note that in the TDM formulation, the agent receives zero reward in intermediate timesteps; it only receives a non-zero reward signal at the end of all ten timesteps, once all parameter outputs are accumulated. For more information on our TDM formulation, please refer to Appendix 5.3.

**Workload and Optimization Objectives.** From a DRAM perspective, a workload is a set of memory read/write accesses. Thus, we pick two extreme access patterns in this context: a streaming memory access trace and a random memory access trace. These are representative of real-world workloads. For instance, dense matrix-matrix multiplication and convolution operations would result in streaming access. Likewise, operations like pointer chasing and sparse matrix-matrix multiplication will result in random memory access. For more details, we refer the readers to Appendix 5.1.

Our goal is to design a memory controller for three different objectives – (a) Low latency, (b) Low Power, (c) Low latency and low power. These are realistic objectives depending on the target domain. For instance, in an embedded use case, achieving low power might be critical, whereas achieving low latency in a high-performance computing domain is critical. We want to compare the performance of single-agent RL with multi-agent RL formulations against these optimization objectives.

**DRAM-Gym Environment.** We focus our empirical study on DRAMSys [Steiner et al., 2020], which is a fast transaction-level simulator for modeling the performance of the DRAM memory system. DRAMSys takes memory traces as input and exposes several parameters in the memory subsystem, such as address mapping, memory controller parameters, and DRAM technology. Then, the simulator outputs the latency and power for that memory traces for a given parameter selection in the DRAM system.To evaluate the effectiveness of different RL methods, including multi-agent RL, we encapsulate DRAMsys into a Gym environment [Brockman et al., 2016], which provides a common interface for RL and MARL algorithms. We next describe the components of the RL formulation used in our experiments.

*Observations.* In this version of the DRAM-Gym environment, the observation is a tuple of `<latency, power, energy>`. This observation is similar to a multigrid world environment where the objective is to reach a specific coordinate. In our case, the coordinates are an n-dimensional parameter space where each combination of the parameter space will have a corresponding `<latency, power, energy>`. We aim to find the specific coordinates in the n-dimensional space (i.e., memory controller parameters) that satisfy the optimization objectives.

*Actions.* The considered action space corresponds to the parameters in the memory controller. The action space is diverse; some actions are categorical variables (e.g., `Page policy`), and some are integers (e.g., `Request Buffer Size`). For more information on the parameter space for the memory controller we explore in this work, we refer the readers to Appendix 5.2 (Table 1).

*Rewards.* We use similar reward structure for the objective of minimizing power and latency. For minimizing both, we use the following reward function

$$r_x = \frac{X_{\text{target}}}{|X_{\text{target}} - X_{\text{obs}}|}\,, \tag{1}$$

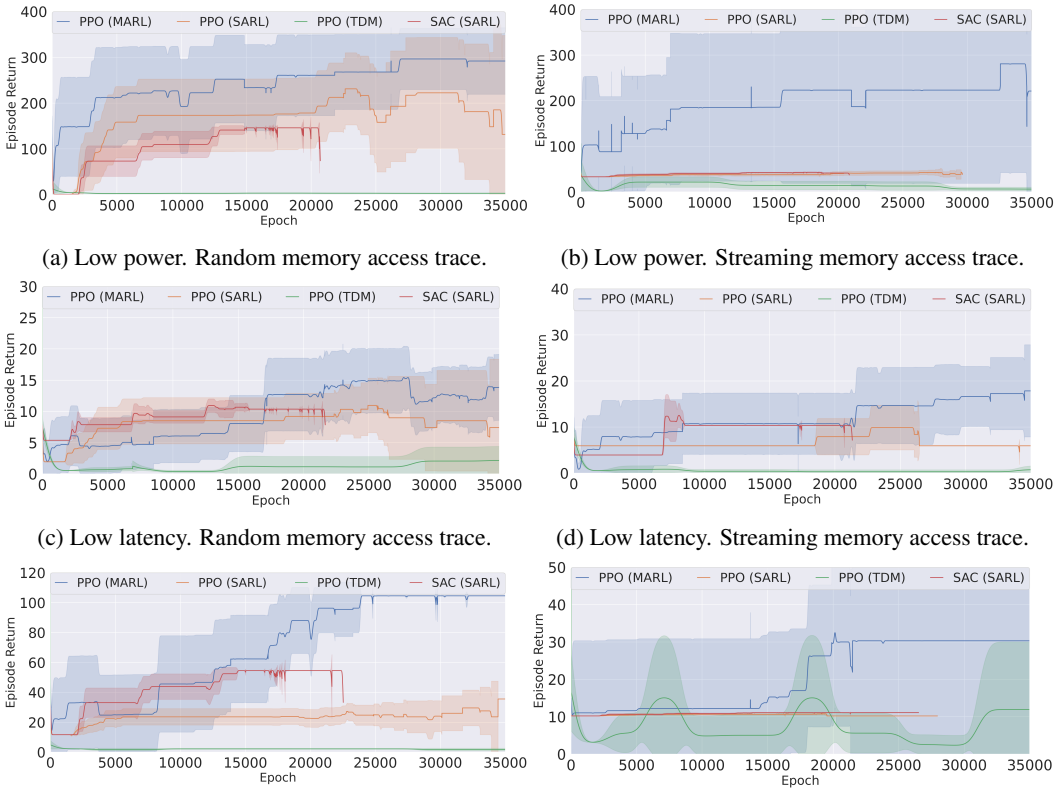
where $X_{\text{target}}$ is the target power/latency and $X_{\text{obs}}$ is the currently-achieved value. For the joint objective of minimizing both power and latency, we take the product of $r_{\text{joint}} = r_{\text{power}} * r_{\text{latency}}$.

## 3 Results

This section describes our evaluation of the MARL formulation for designing a domain-specific DRAM memory controller for two different workload traces and three different optimization objectives. It also compares against single-agent RL formulations.Our goal is to design a custom memory controller that achieves low power, low latency, and a joint objective of achieving low power and latency. Our results demonstrate that MARL formulations consistently outperform single-agent RL formulations across both workloads and all three optimization objectives.

We compare the MARL performance for three different design objectives: low power, low latency, and joint optimization of low power and low latency based on mean episodic return. Given the rewards specified by Eq. 1 are closely tied with the power, latency, and energy, a higher mean return signifies a memory controller design that achieves better architectural performance and energy efficiency.

*Low Power.* Figure 2a and Figure 2b compares the performance of MARL formulation with three single-agent RL baselines for designing a low-power DRAM memory controller for random memory access trace (Figure 3a) and streaming memory access trace (Figure 3b). The reward function of all the agents is given by Eq 1. PPO_TDM achieves a mean episode return of $\approx 4.29$. Likewise, for PPO and SAC agents, the mean episode reward is 48.44 and 146 compared to MARL's mean episode reward of $\approx 300$. Thus, MARL formulation achieves $2\times$, $6\times$, and $60\times$ better mean episode return




(a) Low power. Random memory access trace.

(b) Low power. Streaming memory access trace.

(c) Low latency. Random memory access trace.

(d) Low latency. Streaming memory access trace.

(e) Joint optimization of random memory access.

(f) Joint optimization of streaming memory access.

Figure 2: (a),(c), and (e) Mean episode return for `random memory access` trace workload for low power target, low latency target, and jointly optimize for low power and latency. (b),(d), and (f) Mean episode return for `streaming memory` access trace workload for low power target, low latency target, and jointly optimize for low power and latency. Each RL formulation was run for five seeds each and the error bars signifies 95% confidence intervals. PPO (MARL) is the MARL formulation. PPO (SARL), ppo (TDM), and SAC (SARL) are the single-agent formulation for the same task. Appendix A has more information about single-agent RL formulations.

compared to PPO, SAC, and vanilla PPO RL formulations. A mean return of $\approx 300$ corresponds to a memory controller design that dissipates 1 Watt.

*Low-Latency.* Figure 2c and Figure 2d shows the performance of MARL formulation in comparison with single-agent RL formulations for designing a low latency DRAM memory controller for random and streaming memory access traces. For the low-latency objective, our evaluation shows that MARL formulation, on average, achieves $\approx 1.5 \times$ to $2\times$ more mean episodic return compared to other single-agent RL formulations.

*Joint Optimization of Low Power and Low Latency.* Figure 2e and Figure 2f shows mean episodic return for MARL formulation with other single-agent RL formulations. MARL formulations consistently achieve $\approx$ 2-2.5$\times$ mean episodic return compared to other single-agent formulations.

Our exhaustive evaluation across five seeds across different memory access traces and optimization objectives shows that MARL outperforms single-agent RL formulations.

## 4 Discussion and Future Work

This work presents the first MARL formulation for the hardware architecture search problem. We show that complex interconnected systems have independent parameters that can be mapped to decentralized agents working together to achieve a target. Our exhaustive evaluation shows that our MARL approach consistently outperforms several single-agent RL formulations for hardware architecture searches for DRAM memory controllers for different workloads.

Despite the performance of our method, we must understand the pitfalls regarding resource constraints. While assigning one agent per parameter, as we explored in this paper, is promising, as system complexity increases this approach will also incur hardware resource overheads (each agent has its independent neural network policy). Our future work is centered around the following directions.

**Increase Computer System Complexity.** We plan to expand the scope of architecture DSE problems from DRAM systems to custom hardware accelerators or SoC designs. Increasing the complexity of the architecture DSE from DRAM to SoC designs will also result in a much larger design (million to billion times larger design space) to evalaute the efficiacy of our approach. In addition, we want to explore how to systematically apply MARL formulations to computer architecture DSE problems as we continue to validate our MARL formulations.

**Non-RL Baselines.** In this paper, we propose decentralized multi-agent RL as an alternative single-agent RL for architecture DSE. In the future, we would like to benchmark the performance of our approach on other ML baselines such as Bayesian optimization [Reagen et al., 2017], evolutionary algorithms [Bäck and Schwefel, 1993], and other bio-inspired multi-agent algorithms such as ant colony optimizations [Goss et al., 1989]. Applying popular non-RL baselines will allow us to holistically evaluate various trade-offs in applying decentralized MARL algorithms for architecture DSE.

**Improve MARL formulation.** We would like to systematically understand how to scale multiple agents for a given architecture DSE problem. For instance, Figure 1b, some parameters play less roles in a target objective. We believe we can use this information to prune the number of agents, thereby improving the overall performance of MARL formulation. Alternatively, we can cluster parameters belonging to the same subsystem and assign them to the same agent.

**Generalizability**. MARL, by default, provides us with a way to push the boundary on generalization. However, as we scale the complexity of the computer system, we would like to see how a group of agent's policies (let's say memory) can be reinitialized as we evaluate different workloads or a completely different computer system but having the same sub-system (e.g., memory sub-system).

## 5 Acknowledgements

The authors would like to thank Yanqi Zhou (Google Research, Brain Team) and Douglas Eck (Google Research, Brain Team) for reviewing this manuscript internally during the approval process. The authors would also like to thank Niko Grupen (Google Research, Brain Team) for his helpful documentation on the hyperparameter sensitivity study in ACME. The authors also thank Asma Ghandeharioun, Andrei Kapishnikov, and Yannick Assogba for making the multiagent ACME codebase available. This research is based upon work supported in part by the Google Student Researcher program and the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via 2022-21102100013. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

## References

M. Baak, R. Koopman, H. Snoek, and S. Klous. A new correlation coefficient between categorical, ordinal and interval variables with pearson characteristics, 2018. URL `https://arxiv.org/abs/1811.11440`.

Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

Adrián Barredo, Jonathan C. Beard, and Miquel Moretó. Poster: Spidre: Accelerating sparse memory access patterns. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 483–484, 2019. doi: 10.1109/PACT.2019.00056.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

William J. Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. *Commun. ACM*, 63(7):48–57, jun 2020. ISSN 0001-0782. doi: 10.1145/3361682. URL https://doi.org/10.1145/3361682.

Christian Schröder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *CoRR*, abs/2011.09533, 2020. URL https://arxiv.org/abs/2011.09533.

Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, pages 365–376, 2011.

Simon Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques Marie Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alexander Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Çaglar Gülçehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *CoRR*, abs/2006.00979, 2020. URL https://arxiv.org/abs/2006.00979.

Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. Self-optimizing memory controllers: A reinforcement learning approach. *SIGARCH Comput. Archit. News*, 36(3):39–50, jun 2008. ISSN 0163-5964. doi: 10.1145/1394608.1382172. URL https://doi.org/10.1145/1394608.1382172.

Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 622–636, 2020. doi: 10.1109/MICRO50266.2020.00058.

Ting-Ru Lin, Drew Penney, Massoud Pedram, and Lizhong Chen. A deep reinforcement learning framework for architectural exploration: A routerless noc case study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 99–110. IEEE, 2020.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017. URL https://arxiv.org/abs/1706.02275.

Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. A case for efficient accelerator design space exploration via bayesian optimization. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2017.

Robert R Schaller. Moore's law: past, present and future. *IEEE spectrum*, 34(6):52–59, 1997.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. Autockt: Deep reinforcement learning of analog circuit designs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 490–495. IEEE, 2020.

Jong Hoon Shin, Ali Shafiee, Ardavan Pedram, Hamzah Abdel-Aziz, Ling Li, and Joseph Hassoun. Design space exploration of sparse accelerators for deep neural networks. *CoRR*, abs/2107.12922, 2021. URL https://arxiv.org/abs/2107.12922.

Lukas Steiner, Matthias Jung, Felipe S Prado, Kirill Bykov, and Norbert Wehn. Dramsys4.0: A fast and cycle-accurate systemc/tlm-based dram simulator. In *International Conference on Embedded Computer Systems*, pages 110–126. Springer, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *The 57th Design Automation Conference (DAC)*, 2020.

Amir Yazdanbakhsh, Aviral Kumar, Kevin Swersky, Milad Hashemi, and Sergey Levine. Data-driven offline optimization for architecting hardware accelerators. In *International Conference on Learning Representations 2022*, 2022.

Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen, and Yi Wu. The surprising effectiveness of MAPPO in cooperative, multi-agent games. *CoRR*, abs/2103.01955, 2021. URL `https://arxiv.org/abs/2103.01955`.

Dan Zhang, Safeen Huda, Ebrahim Songhori, Kartik Prabhu, Quoc Le, Anna Goldie, and Azalia Mirhoseini. A full-stack search technique for domain optimized deep learning accelerators. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, page 27–42, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392051. doi: 10.1145/3503222.3507767. URL `https://doi.org/10.1145/3503222.3507767`.

# Appendix

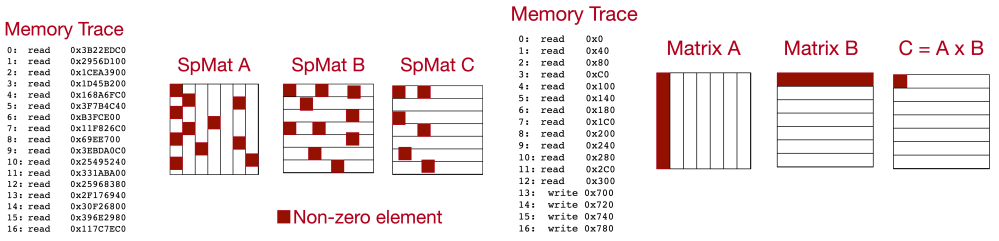
## 5.1 DRAM-Gym and Example Workloads

We evaluate two memory access traces: streaming access and random memory access. Both choices are representative of real-world workload. For instance, convolution (without zero-skipping) results in dense matrix arithmetic, which involves reading each element from two input matrices in a fixed stride pattern, thus resulting in a streaming access pattern, as shown in Figure 3b.

Likewise, the random memory access trace denotes system-level optimizations that exploit sparsity [Shin et al., 2021, Barredo et al., 2019]. For example, non-linear operations such as ReLU cause a lot of zero elements in the output. For latency and energy savings, multiplication with zero elements is avoided, and only the indices of the non-zero elements are stored and accessed. Since it is hard to determine the exact indices of non-zero elements in a matrix during runtime, multiplying two sparse matrices will result in a non-deterministic random access pattern, as shown in Figure 3a.

## 5.2 DRAM Memory Controller Parameters

We focus on the memory controller design within the main memory (DRAM) system. Prior work has shown that memory controller design plays a significant role in latency and energy incurred during data transfer from main memory to compute engine (e.g., cores). To model the DRAM system, we use




(a) Random memory access trace. (b) Streaming memory access trace.

Figure 3: (a) Random memory access workload trace. Sparse matrix-matrix arithmetic results in random memory access. (c) Streaming memory access trace. Dense matrix-matrix arithmetic results in a streaming address pattern.

Table 1: DRAMSys Memory Controller Parameters. Note that a mixture of numerical and categorical parameters are learned for the memory controller.

| Parameter | Min | Max | Step | Type |
|---|---|---|---|---|
| RefreshMaxPostponed | 1 | 8 | 1 | Numerical |
| RefreshMaxPulledin | 1 | 8 | 1 | Numerical |
| RequestBufferSize | 1 | 8 | 1 | Numerical |
| MaxActiveTransactions | 1 | 128 | $2^x$ | Numerical |
| **Parameter** | **Values** | | | **Type** |
| PagePolicy | Open, OpenAdaptive, Closed, ClosedAdaptive | | | Categorical |
| Scheduler | Fifo, FrFcfsGrp, FrFcfs | | | Categorical |
| SchedulerBuffer | Bankwise, ReadWrite, Shared | | | Categorical |
| Arbiter | Simple, Fifo, Reorder | | | Categorical |
| RespQueue | Fifo, Reorder | | | Categorical |
| RefreshPolicy | NoRefresh, AllBank | | | Categorical |

DRAMSys [Steiner et al., 2020]. DRAMSys is a fast, cycle-accurate simulator that exposes DRAM architectural parameters spanning from address mapping, memory controller, memory specification (DDR3/DDR4, etc.), memory technology (HBM, DDR, etc.), and thermal properties. In this work, we focus on parameters within the memory controller and use MARL and other single-agent RL formulations to evaluate the best memory controller parameter for a given application.

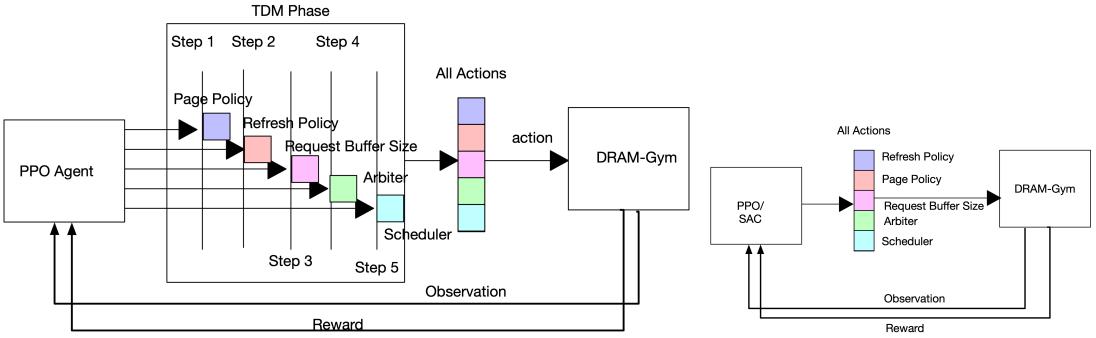
The list of parameters within the memory controller is tabulated in Table 1. Varying these memory controller parameters controls how to service each address request from the compute engine. Hence, the policies and parameters inside the memory controller affect the runtime latency and power for the overall application.

### 5.3 RL Algorithms and its Hyperparameters

To remove statistical noise, we run a wide hyperparameter sweep for each evaluation involving five seeds, learning rate, and entropy values. We evaluate that trained policy at an interval of 100 training steps and report the mean episode return.

Below, we describe the hyperparameters we used for our MARL agents, TDM agents, PPO, and SAC agents, respectively.

**MARL.** In our decentralized MARL formulations, we assign one agent for one memory controller parameter we control, as shown in Figure 1a. Each agent gets the same observation. All agents take




(a) TDM Formulation.

(b) PPO and SAC formulations.

Figure 4: (a) TDM formulation for DRAM-Gym. In TDM, at each step the agent takes action for one parameter. In TDM phase, it accumulates all the actions together and sends it to the environment. At the end of the step, it gets a non-zero reward. (b) PPO and SAC single-agent formulations.

Table 2: The hyperparameters used in PPO and SAC agents. For the MARL formulation, each agent is an decentralized PPO agent with independent policy. Likewise for TDM, there is one PPO agent.

(a) TDM/PPO/MARL Hyperparameters

| Hyperparameter | Value |
|---|---|
| batch_size | 128 |
| discount | 0.99 |
| learning_rate | {1e-5, 2e-5, 2e-4, 1e-4} |
| adam_epsilon | 1e-5 |
| num_minibatch | 32 |
| unroll_length | 16 |
| num_epochs | 5 |
| clip_value | False |
| clipping_epsilon | 0.2 |
| gae_lambda | 0.95 |
| entropy_cost | 0.01 |
| value_cost | 1.0 |
| max_gradient_norm | 0.5 |
| prefetch_size | 4 |
| variable_update_period | 1 |

(b) SAC Hyperparameters

| Hyperparameter | Value |
|---|---|
| batch_size | 256 |
| discount | 0.99 |
| learning_rate | {1e-5, 2e-5, 2e-4, 1e-4} |
| reward_scale | 1 |
| n_step | 1 |
| entropy_coefficient | None |
| target_entropy | 0.0 |
| tau | 0.005 |

simultaneous independent actions and get a global reward (Eq 1). Each decentralized agent in our MARL is a PPO agent.

Table 2 tabulates the hyperparameters we use for MARL agents.

**Time Division Multiplexing (TDM).** Figure 4a shows the details of the TDM formulation we use for DRAM-Gym. As an illustration, a single PPO agent at step 1 predicts the value of `Page Policy` parameter. Step 2 predicts the value for `Refresh Policy`, and so on. The number of steps in the TDM phase corresponds to the number of parameters. The agents get a zero reward for each step in the TDM phase. Once all the actions are accumulated, the actions are passed to the DRAM-gym environment. The agent gets a non-zero reward, and the next TDM iteration starts. The hyperparameters we use for the TDM phase are tabulated in Table 2.

**PPO and SAC Single Agent Fomulations.** Figure 4b shows the single agent RL formulation we use for DRAM-Gym. We use two RL algorithms, namely PPO and SAC. At each time step, a single agent (PPO/SAC) predicts all the parameter values of the memory controller parameters. The agent receives a reward based on Eq 1. The hyperparameters we use for the TDM phase are tabulated in Table 2.