2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

An Actor-Critic Architecture for Community Detection Ablation Study

Rafael Henrique Nogalha de Lima, Aurélio Ribeiro Costa, Thiago de Paulo Faleiros, Célia Ghedini Ralha
Exact Science Institute - Computer Science Department - University of Brasília - Brazil
rafaelnogalha@gmail.com, arcosta@gmail.com, thiagodepaulo@unb.br, ghedini@unb.br

***Abstract*—This article conducts an ablation study of the Actor-Critic Architecture for Community Detection (AC2CD). The AC2CD uses Deep Reinforcement Learning (DRL) and Graph Attention Networks (GAT). Our ablation study method adheres to the principles of explainable artificial intelligence, focusing on assessing performance factors, including execution time, memory usage, and GPU utilization. We carried out experiments using two real-world datasets: Email-Eu-Core (EC), an email network among members of a European research institution (comprising 1,005 nodes, 25,571 edges, and 42 communities) available through the Stanford Snap Project, and a High School contact and friendship network (HS) in Marseilles, France, from December 2013 (comprising 329 nodes, 45,047 edges, and nine communities), obtainable from the Socio Patterns Website. We evaluated performance while considering three hyperparameters: learn_rate (LR), batch_size (BS), and n_games (NG), varying them at 10%, 30%, 50%, and 70%. The LR of 70% yielded optimal results with execution time for both EC and HS datasets. Furthermore, a BS of 70% indicated an ideal balance between execution time, GPU usage, and memory consumption for the HS dataset.**

***Index Terms*—Ablation Study, AC2CD, Hyperparameters**

I. Introduction

Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) refer to learning how to make decisions sequentially while being influenced by the environment, becoming mature in the past years [1]. In short, the RL goal is to map situations to actions that maximize a numerical reward signal indicating how well the agent performs tasks. Agents learn through trial and error, adjusting actions to achieve the highest possible reward. And DRL integrates deep learning into RL techniques to train an agent.

Given the advancement in research and the diverse applications of ML, specifically RL, in various fields, including scientific and commercial domains, it becomes imperative to understand the impact of selecting specific components and parameters for developing an ML system, given its complexity. Therefore, a compelling approach to address this issue is to conduct an Ablation Study (AS) [2]. As presented in [3], the AS technique is a scientific examination of ML systems to gain insight into the effects of code blocks on performance. And because of that, the method of AS has gained significant attention in the field of ML in recent years.

The AS findings inform future research on optimizing actor-critic architectures and potentially lead to the development of automated hyperparameters (HP) tuning techniques. The actor-critic is a Temporal Difference (TD) version of policy gradient with two networks: actor and critic. The actor decides which action to take, and the critic informs the actor how good the action was or how to adjust it. Investigating the optimization process at the impact of learning strategies with HP contributes to the broader understanding of DRL models and their transformation towards based Explainable Artificial Intelligence (XAI) [4].

Therefore, the main objective of this work is to present an AS using an actor-critic architecture developed upon DRL and Graph Attention Networks (GAT), called the Actor-Critic for Community Detection (AC2CD) [5]. GAT is significant for ML research, but it has received low levels of attention. Thus, the AS analyzes the algorithm performance by empirically modifying specific HP when executing real-world datasets. The AC2CD uses as datasets an email network between members of a European research institution, Email-Eu-Core (EC), a High School (HS) contact and friendship network in Marseilles/France in December 2013, the BlogCatalog3, the Flicker and the Youtube2. In this AS, we used EC and HS as presented in Section IV.

This work contribution presents an AS with the AC2CD architecture, including execution time, memory, and GPU usage to assess performance. The insights gained contribute to the ongoing efforts to enhance the efficiency and effectiveness of DRL algorithms in real-world applications.

The rest of the manuscript presents in Section II preliminaries with an overview of concepts, in Section III related work, in Section IV the experimental AS method, in Section V the results, and in Section VI conclusion and future work.

II. Preliminaries

Section II-A presents an overview of AI concepts with DRL, and XAI. Section II-B offers community detection aspects. Section II-C presents the actor-critic architecture applied to the community detection problem with the AC2CD algorithm. Finally, Section II-D presents AS definition and techniques.

A. AI Overview

For [6], DRL is an RL approach combined with deep learning employed when decisions become too complex for RL alone. A neural network estimates states instead of mapping all possible solutions, allowing for a more manageable solution space in the decision-making process.

With the growth of ML approaches, various domains of knowledge have benefited. However, systems with AI have

978-0-7381-4408-5/23/$31.00 ©2023 IEEE

become complex and hard to understand and explain. As a result, a new approach to AI-based systems has emerged to provide explainability to human users, highlighting the strengths and weaknesses of the algorithm and conveying an understanding of how it will behave in the future. According to [7], XAI enables greater transparency and interpretability in complex AI systems allowing users' trust and permitting humans to make informed decisions while effectively cooperating with such systems. XAI bridges the gap between the black-box nature of traditional AI and the human need for comprehensibility by providing explanations for algorithmic decisions. That enhances the usability and ethical considerations of AI applications across various domains.

B. Community Detection

Community detection is one of the fundamental problems in network analysis, belonging to the field of complex network studies. According to [8], the community detection technique is characterized by having a community structure, where the nodes in the network are grouped into sets such that each set of nodes is densely connected. For [9], community detection is the process of identifying relevant communities in a network that evolves as in a dynamic network. Community detection is vital to understanding the structure of complex networks. Community detection techniques are helpful for social media algorithms to discover people with similar opinions, functions, purposes, and shared interests significant to scientific inquiry and data analytics. There are classic methods of community detection using spectral clustering [10] and statistical inference [11]. However, such methods drop out, as deep learning techniques demonstrate an increased capacity to handle high-dimensional graph data with impressive performance.

C. Actor-critic

Actor-critic is a TD learning method representing the policy function independent of the value function. The policy function returns a probability distribution over possible agents' actions based on the provided state or the agents' strategy to achieve a goal. On the other hand, the value function determines the expected return for an agent starting in a particular state and continually acting under a specific policy [1], [12]. In the actor-critic learning method, the actor decides which action to take. The critic provides feedback to the actor on the quality of the action and how it can be adjusted to achieve the goal [13]. In short, the actor-critic is a hybrid architecture combining value-based and policy-based methods that help to stabilize the training by reducing the variance. It provides a solution to reducing the RL algorithm variance, training agents faster and better.

The AC2CD is a DRL-based architecture with a GAT approach. GAT are novel neural network architectures operating on graph-structured data, leveraging masked self-attention layers [14]. It is employed to find an optimal community structure in a dynamic social network while also serving as a learning component to select actions and improve the value function [5]. Experimental work indicates that AC2CD copes well with dynamic real-world social networks. Nevertheless, the performance of such complex architecture motivates AS approach to enhance performance to evaluate the growing size of dynamic social networks.

Algorithm 1 presents the AC2CD for community detection, including the input (i.e., Input:) with *Dataset* and the HP. The manipulation of *Dataset* is made by *DataManip(Dataset)*. The method *Agent(Hp)* create the agent according to *Hp* and consequently makes the actor, the Critic networks, and the experience memory store the last episodes of execution. Inside the loop, *line 5* to *line 12*, while not in a terminal state, the agent chooses an action based on the observations and executes a step returning a new observation, the reward, and a flag indicating that the new state is terminal. After the learning process, the model file is ready to infer the community assignment for nodes in a network, and the output is the score stored in the *score_history* variable.

Algorithm 1: Community Detection in AC2CD

```
   Input: Dataset
   Input: Hp
1  node_emb, edge_index ← DataManip(Dataset);
2  env ← GATEnv(node_emb, edge_index);
3  agent ← Agent(Hp);
4  score_history ← [];
5  while n < Hp.max_iter do
6      obs ← env.reset();
7      while not done do
8          action, prob, val ← agent.choose_action(obs);
9          new_obs,reward,done ← env.step(action);
10         agent.remember(obs, action, reward,prob,val);
11         if n % Hp.train_interval == 0 then
12             agent.learn();
```

D. Ablation Study

The first idea of AS comes from speech recognition studies [15]. Although not a new idea, it is a relatively young AI research theme [4]. AS is defined by [16] as a scientific method that involves highlighting or removing individual or blocks of components from a system to prove and understand which aspects of a system are vital through statistical analysis. Using statistics and analyzing the results obtained from AS, it is possible to gain insights into the relative importance of the parameters of architecture or model. With these insights, improving systems' design, optimization, and interpretability is possible. AS is a valuable tool for discovering the component's influence in ML systems. Through statistical analysis, it is possible to enhance the interpretability of ML approaches.

The use of AS in XAI systems becomes interesting, considering its complementarity to understanding AI systems. The AS aims to understand the importance of parameters and code blocks in an architecture or model [4]. This study enables identifying and quantifying the influence of various components on an algorithm, model, or architecture, leading to a better understanding of the underlying mechanisms.

This understanding is crucial for building trust and ensuring transparency in AI systems.

III. Related Work

This section presents related work of AS, DRL, and actor-critic with publications from 2018 to 2023. Table I presents an outline including AS, DRL, actor-critic, and GAT aspects.

TABLE I
RELATED WORK OUTLINE.

Reference	AS	DRL	Actor-critic	GAT
Fan et al. (2023) [17]	✓	✓	✓	
Naqvi & Anggorojati (2022) [18]	✓	✓		
Ye et al. (2022) [19]	✓	✓	✓	
da Silva Filho et al. (2022) [20]	✓	✓		
Hessel et al. (2018) [21]	✓	✓		
This study	✓	✓	✓	✓

The authors in [17] present a new approach using DRL and actor-critic for a multi-agent system that analyzes and simulates an environment with multiple intelligent agents across various domains. Additionally, the authors conduct an AS to assess the effectiveness of the innovative components in the proposed method. The results show that each actor-critic algorithm component is indispensable for good interception performance, including success rate, good reward, and interception steps.

In [18], the authors explore the utilization of DRL for congestion control in cellular network settings. Congestion control uses algorithms responsible for regulating the data transmission rate in a network to prevent congestion. Using the policy gradient method, the author employs an AS to identify the component that influences the algorithm. With the AS, the authors remove or modify parameters to analyze the impact of the changes on the algorithm's performance. In conclusion, a higher reward for the method presented is only sometimes related to better networking performance.

The authors in [19] focus on the popularity of multi-agent DRL demand for large-scale real-world tasks, which hamper the models' low sample efficiency and the high data collection cost. An AS is used to investigate, validate and understand the contribution of each component in multi-agent actor-critic methods. The authors propose PEDMA, a plugin unit for multi-agent DRL that consists of three techniques: parallel environments to accelerate the data acquisition, experience augmentation utilizing the permutation invariance property to reduce the cost of acquiring data, and delayed updated policies to improve the data utilization efficiency. Experiments on three benchmark tasks show that the multi-agent actor-critic model trained with PEDMA outperforms state-of-the-art algorithms.

The authors in [20] discuss the learning-to-optimize method for automatically optimizing algorithms from data instead of using traditional HP tuning. The focus is on learning global optimization by DRL. It provides a direct framework to understand an optimizer to deal with the exploration-exploitation dilemma. The applied techniques improved stability and generalization. The authors conducted AS to investigate the significance of learning strategies and components concerning this optimization.

In [21], the authors perform an AS to understand the components and parameters contribution in the Deep Q-Network (DQN) algorithm. DQN utilizes DRL to address learning challenges in complex and high-dimensional environments. Each ablation phase removes or changes an algorithm component, and the algorithm's performance is analyzed. The authors propose Rainbow to combine improvements in DRL. In experiments, six integrated DQN algorithm extensions are empirically studied. The results show that the combination provides state-of-the-art performance on the Atari 2600 benchmark considering data efficiency and final performance.

Note that this work is the only one focusing on GAT as a novel convolution-style neural network architecture. GAT is the most popular approach to the community detection problem, but it has received comparatively low levels of attention in performance comparative studies. In addition, the actor-critic architecture disseminated in DRL applications is highly resource-demanding. Thus, the motivation of our AS research is to assess the effectiveness of HP through the AC2CD case study with GAT and actor-critic.

IV. Experimental Method

Figure 1 presents the experimental method with three steps. The first step includes the datasets and baseline definitions. The EC dataset, available on the Snap Project website[1] and the HS, available on the Socio Patterns website[2] are used as input to AC2CD. The second step includes HP variation during the AC2CD executions (*tunning*). Finally, in the third step, we use the AS to observe the importance of the selected HP. The effect analysis focuses on the execution time, GPU, and memory usage.

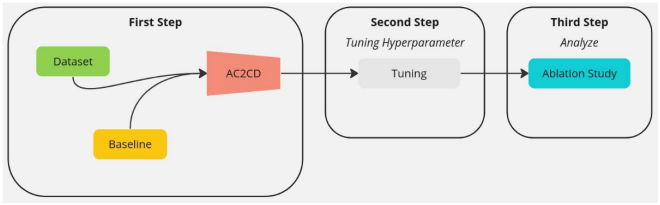

Fig. 1. Experimental method diagram.

The EC is a directed network representing an email network between members of a European research institution. According to [22], the network is formed by an edge (u, v), where u represents the person who sends at least one email to v. The communities in the dataset represent the departments in the organization. There are $1,005$ nodes, $25,571$ edges, and 42 communities, with the longest path being seven, and the average clustering coefficient is 0.3994.

The HS comprises a directed network of contacts and friendship relations between students in a high school in

[1] https://snap.stanford.edu/data/email-Eu-core.html
[2] http://www.sociopatterns.org/datasets/high-school-contact-and-friendship-networks/

Marseilles, France, in December 2013. According to [23], the network is formed by lines with the form (ij), meaning that the student i reported a friendship with student j. And is formed by a metadata file where each line has the form $(IDiCiGi)$, meaning that class Ci and gender Gi of the student has IDi. There are 329 nodes, 45047 edges and 9 communities.

The experiments use a computer with a CPU Intel® Xeon Gold 5220R with 48 cores, 187GB of RAM, and two GPU NVIDIA® V100S. The operating system used is Ubuntu with external libraries provided by the Conda project.[3]

A. Definitions

The AS setup uses *I*, the set of previously defined percentage variations (10%, 30%, 50%, 70%). These percentage variation values were based on their coverage and the comprehensiveness they provided with the conducted tests. We defined three HP of *H* for the experiments $(learn\_rate, batch\_size, n\_games)$. The *learn_rate* (LR) determines the extent to which an agent learns from each sample in the environment [24]. The *batch_size* (BS) represents the number of samples propagated during the training session [25]. Lastly, *n_games* (NG) defines the number of episodes the agent will process.

Definition 1. HP refer to parameters set before the model is trained, rather than estimating from learning, as they define the model architecture [26]. They are used to configure an ML model and specify the algorithm to minimize the loss function, for example.

The HP and their respective values were chosen according to the suggestion of the AC2CD architecture author, considering the previously analyzed influence of each parameter on the developed algorithm. Therefore, the baseline values for the execution of the experimental method regarding each hyperparameter of *H* are 40, 40, and 100 for the LR, BS, and NG, respectively.

Definition 2. Baseline is the reference for a particular ML study [4]. In our case, it refers to the set of executions with the default values of the HP used to compare the variations of the selected HP.

After defining the values of HP for the baseline, the AC2CD architecture with the memory and GPU profiler using Scalene proceeds.[4] The AS execution starts, and the matplotlib[5] tool to generate graphics is used for GPU and memory analysis. This experiment aims to determine a set of HP value for each *H* that minimize GPU and memory consumption for the AC2CD while achieving maximum speed. As shown in Figure 2, Scalene is chosen due to its superior performance compared to other well-known profilers. The results indicate its effectiveness in slowing down the program, profiling memory and GPU usage, and providing system time analysis. Additionally, a *.json* file generated by Scalene allows for line-level and function-level profiling, offering information about specific functions and lines of code, as documented in [27].

Profiler	Slowdown	Lines or Functions	Unmodified Code	Threads	Multi-processing	Python vs. C Time	System Time	Profiles Memory	Python vs. C Memory	GPU	Memory Trends	Copy Volume	Detects Leaks
CPU-only profilers													
pprofile (stat.)	1.0×	lines	✓	✓	-	-	-	-	-	-	-	-	-
py-spy	1.0×	lines	✓	✓	✓	-	-	-	-	-	-	-	-
pyinstrument	1.7×	functions	✓	-	-	-	-	-	-	-	-	-	-
cProfile	1.7×	functions	✓	-	-	-	-	-	-	-	-	-	-
yappi wallclock	3.2×	functions	✓	✓	-	-	-	-	-	-	-	-	-
yappi CPU	3.6×	functions	✓	✓	-	-	-	-	-	-	-	-	-
line_profiler	2.2×	lines	-	-	-	-	-	-	-	-	-	-	-
Profile	15.1×	functions	✓	-	-	-	-	-	-	-	-	-	-
pprofile (det.)	36.8×	lines	✓	✓	-	-	-	-	-	-	-	-	-
memory-only profilers													
fil	2.7×	lines	-	-	-	-	-	*peak only*	-	-	-	-	-
memory_profiler	≥37.1×	lines	-	-	-	-	-	*RSS*	-	-	-	-	-
memray	4.0×	lines	-	✓	-	-	-	*peak only*	✓	-	-	-	-
CPU+memory profilers													
Austin (CPU+mem)	1.0×	lines	✓	✓	✓	-	-	*RSS*	-	-	-	-	-
Scalene (CPU+GPU)	1.0×	**both**	✓	✓	✓	✓	✓	-	-	✓	-	-	-
Scalene (all)	1.3×	**both**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fig. 2. Comparison of profiler tools.

The executions followed the arrangement $A_{n,p} = n^p$, where $n = card(I \cup baseline\ percentage)$ and $p = card(H)$. This arrangement allowed four variations of percentages according to the I set. Additionally, HP could be varied at 0%, enabling exploring the scenarios' maximum number. Thus, combining with LR and BS varied by 10%, and NG by 0% (using the baseline value). Each dataset had 125 executions and six scenarios, including the baseline for a GPU consumption comparison, memory usage, and execution time. The selection of the six scenarios considered the lowest GPU consumption, memory usage, and execution time. It is important to highlight that the set of HP does not operate individually for GPU consumption, memory, and execution time.

Definition 3. Manual tuning of HP is a technique for adjusting their value. We employed this method to vary four percentages outlined in the I set (10%, 30%, 50%, 70%). We used this approach in conjunction with the conceptualization of AS in Section II-D to determine the optimal configuration of GPU, memory, and time consumption of the three HP specified in the set *H*.

V. Results and Discussion

In this section, we present the AC2CD architecture with *I* variations of 10%, 30%, 50%, 70% using EC and HS datasets, focusing the GPU and memory usage, and run-time execution. The results of the baseline execution for EC were 12.092 GiB for the memory GPU usage, 1.124 GiB for memory consumption, and $1h40m24s$ for run-time execution. And HS was 17.339 GiB for the memory GPU usage, 1.006 GiB for the memory consumption, and $10m26s$ for run-time execution (Figures 3 and 6).

GPU Consumption

Figure 3 presents the best results for GPU usage with EC and HS datasets. Note that GPU consumption with EC (i.e., blue bars) is higher for the baseline (black dotted line with 12.092 GiB), BS(10%) (i.e., 12.025 GiB), and NG(10%) (i.e., 11.999 GiB). For the LR(70%) (i.e., 10.911 GiB), and LR(30%), BS(30%), NG(10%) (i.e., 10.223 GiB), both sets of HP are adequate, but the last is the best result in terms of

[3]Conda Project available at https://docs.conda.io/en/latest/
[4]https://github.com/plasma-umass/scalene
[5]https://matplotlib.org/

GPU consumption. This figure also presents the best results for GPU usage with HS (i.e., green bars). The baseline (red dotted line) is the highest value with 17.339 GiB of GPU consumption, and for this dataset, the best result is LR(70%) and BS(70%) (i.e., 14.123 GiB).

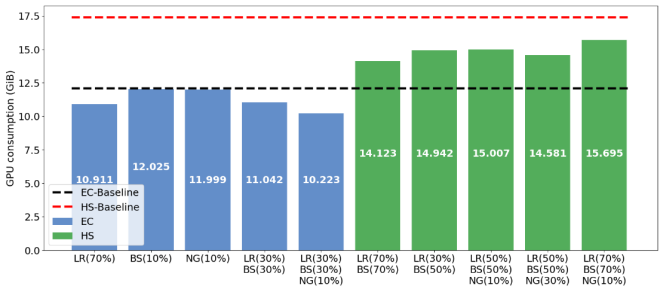

Fig. 3. Best results in terms of GPU consumption with EC and HS datasets.

Furthermore, Figure 4 presents the Scalene interface for LR(30%), BS(10%), and NG(10%) with the EC dataset. The percentage of GPU consumption (i.e., 31.2% corresponds to 10.223 GiB) is displayed using a pie graph in the GPU utilization column (*GPU util.*). The *GPU memory* column presents the code line memory consumption. Finally, the function that consumes the GPU referring to the previously shown code line appears below the *FUNCTION PROFILE* column. Note that the code segment related to agent learning (*Agent.learn*) for each node in the EC is the one that consumes the most GPU in this configuration. In general, this is the code block that consumes most of the AC2CD algorithm. In addition, Figure 5 presents the Scalene interface for the GPU consumption (i.e., 43.1% corresponds to 14.123 GiB) with LR(70%), BS(70%) with the HS dataset, where the *Agent.learn* consumes more GPU memory.

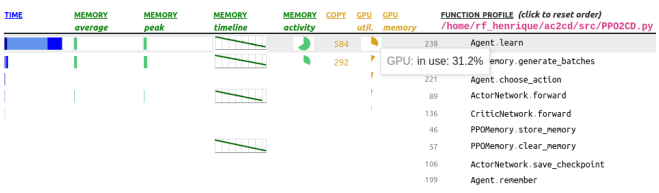

Fig. 4. Scalene interface for LR(30%), BS(10%), NG(10%) with the EC dataset.

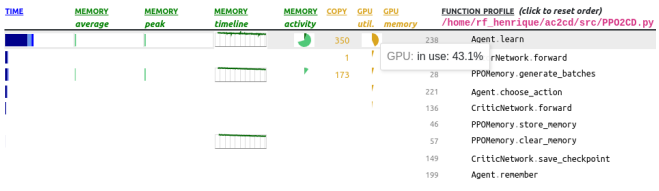

Fig. 5. Scalene interface for LR(70%), BS(70%) with the HS dataset.

What is interesting in the execution of HS is that, despite having fewer communities and being smaller than the EC dataset, it consumes more GPU. That might occur due to the different transformations in the dataset file for EC, where the $torch.LongTensor$ generates the index, transposing data by the generated index, as seen in Listing 1. In contrast, for the HS dataset, both the index and data are derived using the $from\_networkx$ function, which converts a graph to a $torch\_geometric.data.Data$ instance[6] as seen in Listing 2. Another possibility for the high GPU consumption is the use of metadata in HS, which makes the execution of this dataset more complex, requiring more computational power. We might note that even using other datasets, the performance results with LR, BS, and NG HP would be similar since the computational usage is deeply related to the dataset structure.

```
index = torch.LongTensor(
    np.vstack((adj.row, adj.col))
)
data = Data(edge_index=torch.transpose(index,0,1))
enconder = embedding.Encoder(data,index,device=self
    .device)
```

Listing 1. Code execution fragment with the EC dataset.

```
data = from_networkx(graph)
encoder = embedding.Encoder(data,
    data.edge_index,
    device=self.device
)
```

Listing 2. Code execution fragment with the HS dataset.

Memory Consumption

The memory consumption in Figure 6 does not exhibit as much variation comparing the GPU consumption of the EC dataset (Figure 3). We can attribute this behavior to the fact that most memory allocation in the AC2CD is related to tensors GPU stored consumption. Note BS(10%) is the best result of memory consumption with 1.123 GiB.

For the HS dataset, the baseline consumes 1.006 GiB, and for LR(70%), BS(70%) consumes 1.021 GiB, and LR(70%), BS(70%), NG(10%) consumes 1.022 GiB. In relation to LR(50%), BS(50%), NG(30%) the consumption is 1.007 GiB. Finally, LR(30%), BS(50%), and LR(50%) BS(50%), NG(10%) represent the best results for memory consumption, with a consumption of 1.003 GiB.

Run time Execution

Note the EC dataset in Table II, the LR(70%) is the faster execution time with $1h19min23s$. However, the execution with LR(30%), BS(30%), and NG(10%) is the slowest compared to the other five percentage variation executions taking $2h7min35s$. This extense execution time grounds on the influence of NG that determines the number of episodes the agent will process.

Regarding the HS, the execution time is shorter than the EC dataset due to having fewer nodes, edges, and communities. The shortest time is achieved with LR(70%), BS(70%) with $3min25s$, and LR(70%), BS(70%), NG(10%) with $4min08s$ also proved to be a good option. On the other hand, the baseline showed the worst time in the execution comparison with $10min26s$.

[6]https://pytorch-geometric.readthedocs.io/en/latest/

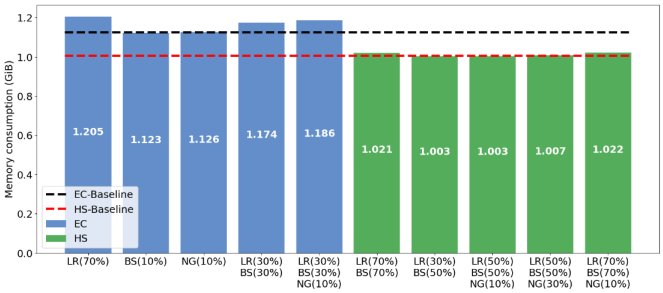

Fig. 6. Best results regarding memory consumption with EC and HS datasets.

TABLE II
RUN-TIME RESULTS FOR EC AND HS DATASETS.

EC hyperparameter	EC Run-time execution	HS hyperparameter	HS Run-time execution
baseline	1h40m24s	baseline	10m26s
LR(70%)	**1h19m23s**	LR(70%) BS(70%)	**3m25s**
BS(10%)	1h25m02s	LR(30%) BS(50%)	4m53s
NG(10%)	1h34m10s	LR(50%) BS(50%) NG(10%)	7m08s
LR(30%) BS(30%)	1h29m23s	LR(50%) BS(50%) NG(30%)	8m05s
LR(30%) BS(30%) NG(10%)	2h7m35s	LR(70%) BS(70%) NG(10%)	4m08s

VI. CONCLUSION

The AS results indicate that DRL architectural structures and HP impact GPU, memory, and run-time execution. The dataset manipulation by the architecture also influences the execution. We suggest the DRL developers attend to simplifying the architecture layers, considering tensors, encoders, and learning algorithms to accelerate execution.

For the EC dataset, the LR(30%), BS(30%), NG(10%), and BS(10%) indicate optimal HP configuration considering GPU and memory usage, respectively, and LR(70%) for run-time execution. Regarding the HS dataset, the LR(70%) and BS(70%) indicate optimal HP configuration for GPU and memory usage and execution time. Although LR(30%), BS(50%), and LR(50%), BS(50%), NG(10%) perform better in terms of memory.

As future work, we consider implementing AS for other AC2CD HP with automatic tuning to different datasets and applications. Furthermore, exploring the influence of AS on the learning phase of DRL and conducting effectiveness and accuracy analyses of the models presents a promising research avenue for advancing studies focusing on XAI.

REFERENCES

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[2] Allen Newel. A tutorial on speech understanding systems. In D. R. Reddy, editor, *Speech Recognition Invited Papers Presented at the 1974 IEEE Symposium*, pages 3–54. Academic Press, CMU, USA, 1975.

[3] Sina Sheikholeslami. Ablation programming for machine learning. Master's thesis, KTH Royal Institute of Technology, School of Electrical Eng. and Comp. Science (EECS), SE-100 44 Stockholm, Sweden, 2019.

[4] Isha Hameed, Samuel Sharpe, Daniel Barcklow, Justin Au-Yeung, Sahil Verma, Jocelyn Huang, Brian Barr, and C. Bayan Bruss. Based-xai: Breaking ablation studies down for explainable artificial intelligence, 2022. arXiv:2207.05566 [cs.LG].

[5] Aurélio Ribeiro Costa and Célia Ghedini Ralha. AC2CD: An actor–critic architecture for community detection in dynamic social networks. *Knowledge-Based Systems*, 261:110202, 2023.

[6] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, et al. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018.

[7] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.

[8] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[9] Rémy Cazabet and Frédéric Amblard. *Dynamic Community Detection*, pages 404–414. Springer New York, New York, NY, 2014.

[10] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Inf. Proc. Sys.*, 14, 2001.

[11] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1), 2011.

[12] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[13] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in Neural Inf. Proc. Sys.*, 12, 1999.

[14] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. arXiv:1710.10903 [stat.ML].

[15] Dabbala Rajagopal Reddy. *Speech recognition: invited papers presented at the 1974 IEEE symposium*. Elsevier, 1975.

[16] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks, 2019. arXiv:1901.08644 [cs.NE].

[17] Dongyu Fan, Haikuo Shen, and Lijing Dong. Switching-aware multi-agent deep reinforcement learning for target interception. *Applied Intelligence*, 53(7):7876–7891, 2023.

[18] Haidlir Naqvi and Bayu Anggorojati. Ablation study of deep reinforcement learning congestion control in cellular network settings. In *Proc. of 25th Int. Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 80–85. IEEE, 2022.

[19] Zhenhui Ye, Yining Chen, Xiaohong Jiang, Guanghua Song, Bowei Yang, and Sheng Fan. Improving sample efficiency in multi-agent actor-critic methods. *Applied Intelligence*, pages 1–14, 2022.

[20] Moésio Wenceslau da Silva Filho, Gabriel A. Barbosa, and Péricles B. C. Miranda. Learning global optimization by deep reinforcement learning. In *Proc. of 11th Brazilian Conference on Intelligent Systems (BRACIS)*, page 417–433, 2022.

[21] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, et al. Rainbow: Combining improvements in deep reinforcement learning. In *Proc. of AAAI Conf. on Artificial Intelligence*, volume 32, 2018.

[22] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proc. of 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, page 817–826, 2009.

[23] Rossana Mastrandrea, Julie Fournet, and Alain Barrat. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PloS one*, 10(9):e0136497, 2015.

[24] Joshua Romoff. *Decomposing the Bellman Equation in Reinforcement Learning*. PhD thesis, School of Computer Science, McGill University, Montreal, Canada, 2021.

[25] Brennan Shacklett, Erik Wijmans, Aleksei Petrenko, Manolis Savva, Dhruv Batra, et al. Large batch simulation for deep reinforcement learning, 2021. arXiv:2103.07013 [cs.LG].

[26] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[27] Emery Berger, Sam Stern, and Juan Pizzorno. Triangulating python performance issues with scalene, 2022. arXiv:2212.07597 [cs.PL].