PHYRE: A Physical Reasoning Framework

Kyra Kerz

***Abstract*— An agent that is set in a uncontrolled and complex environment is able to choose its actions more effectively if it is able to understand and use the knowledge of the physical laws that are inherent to its surroundings. For this purpose PHYRE, a benchmark for physical reasoning, is used to train an agent that is able to solve various classical mechanics puzzles in a 2D physical environment. The proposed agent consists of four neural networks which are used as action-prediction model, in order to predict the best action for a given task. Each generation shapes the outcome from a coarser understanding of the scene to a more specific one and improves the prediction capabilities of the network for finding an action that solves the puzzle. The results are compared against the learning algorithms provided by the PHYRE framework, which our approach outperforms for single tasks but fails to generals over templates.**

I. Introduction

Humans and animals are capable of making assumptions about how objects are interacting with each other based on physical properties like mass, friction and inertia. In order to test this intuition, solving novel physics puzzles has been used to measure the reasoning abilities of humans and animals [1], [2] . The main goal is to gain an understanding about the ability to generalize that intuition for different task setups. Even though research suggests that infants are most likely born with the ability to make sense of physical laws like gravity, this innate reasoning is nowhere near the level of an adult. With five months babies start to understand that solid objects have different properties than liquids [3]. This means that physical reasoning of humans and animals grows more precise the more they learn from their environments.
Our approach is based on the insight, that humans and animals are increasing their ability of physical reason through repetition and exercise. The goal is to develop an agent that is able to use physical reasoning to solve the 2D puzzles provided by the PHYRE framework. For this purpose four neural networks are trained on randomly sampled actions of a specific puzzle. Instead of giving binary labels, indicating an action is able to solve a puzzle or not, a score will be applied as label. The calculation of the score is dependent on properties of the episodes like distance, contact of objects and time, in order to differentiate between actions that get closer to a solution and that have barely an effect for solving the puzzle. After the training phase, a testing phase is started, which is repeated for several generations. Each generation ends with the retraining of the neural networks. For this purpose the training sets are expanded. Newly added actions are generated by using the neural networks to make predictions for test samples. The actions with the best predictions are then appended to the training set. This leads to an improvement over the prediction capability of the neural networks throughout the generations for finding an action that is able to solve the puzzle.

After discussing related work in Section II, we explain the modeling approach in detail in section III. Section IV presents the results of the demonstrations and compares them against the baseline agents of the PHYRE framework.

II. Related Work

In this section related work and state-of-the art work is introduced. This is organized in two parts. First an overview about physical reasoning in general is given and after that work specific to the PHYRE framework is presented.

A. Physical Reasoning

The PHYRE framework is based on research of physical reasoning in regards to intuitive physics, visual reasoning, and learning in computer games and simulated environments. Intuitive physics describes that animals and humans are capable to reason about the physical world using simplified intuitive physical concepts [4], [5]. Probabilistic models [6], [7] or neural networks [8], [9] for physical simulations are used to make predictions about the future state of a simulation e.g whether a 3D stack of blocks will collapse [10]. But in contrast to predict stability of a model or predict the future state based on intermediate states, PHYRE intervenes in the scene by placing an agent in order to get to a desired final state of the scene.
Another aspect comes in the form of visual reasoning. For instance using natural language processing for understanding scenes, benchmarks for this approach are CLEVR [11] and VQA [12]. Another approach is presented by Mao et al. [13]. Their model learns visual concepts of objects that were given a language description, while looking at images and reading paired questions and answers. Other methods are using neural networks to extract subsymbolic image or latent scene representations as for their subsequent reasoning modules [14], [15]. While those approaches require reasoning about the interactions of the objects within the scene, similar to PHYRE, they don't have to make assumptions about the dynamic of the environment, as their scenes are static.
Another aspect is learning in computer games, where an agent has to reason about physics in a complex 2D or 3D environment [16]. One famous example is the Atari game Pong, where the bar has to be placed correctly in order to catch the ball and therefore being aware of the dynamics of the ball [17]. However in the case of computer games models have to learn a behaviour specialized for a certain task, while PHYRE aims to be a benchmark for an agent

that is capable to generalize over a wide range of different tasks.

B. PHYRE Framework

PHYRE is benchmark for physical reasoning that contains a set of simple classical mechanics puzzles in a 2D physical environment. The benchmark is designed to encourage the development of learning algorithms that are sample-efficient and generalize well across puzzles [18]. The tasks involve placing one or two balls in a 2D world, such that the world reaches a state with a particular outcome: object and goal touching for at least three seconds. As baseline algorithms are a random agent, a non-parametric agent with online learning and a Deep Q-network.
From this benchmark other approaches to solve those tasks have emerged. Girdhar et al. [19] are using forward prediction or the ability to forecast what will happen next given some initial world state, is one approach to try to generalize over those puzzles. They find that forward-prediction models improve the performance of physical-reasoning agents. But they also observe that pixel based representations don't necessarily lead to better physical-reasoning performance compared to object based representations. Another method is described by Rajani et al. [20]. The goal of the ESPRIT framework is to find physical interactions of the PHYRE templates and use them to create an agent that is able to generalize over templates. They are collecting a set of natural language descriptions of the episodes and events of the puzzles. The framework learns to generate explanations of how the physical simulation will evolve so that an agent or a human can easily reason about a solution using the language descriptions. A path planning approach is used by Harter et al. [21]. First they predict the path that the objects would follow without placing an action and secondly they predict the path that the target object would have to follow in order to solve the puzzle. The joint information is used to predict the placement of the object that leads to a solved task, but falls behind the baseline DQN provided by the PHYRE framework.

III. Methodology

In this section the implementation details are explained. The overarching idea behind the approach is to split between training and testing phase. First four neural networks are trained on randomly sampled training actions. The labels or scores for those actions were obtained by a score function that takes physical properties of the scene into account. After that they are used to predict the best actions out of given set of unseen test samples. Those predictions are now compared to the actual score when simulating those actions. The best predictions are then used to extend the training samples and retrain the networks. This whole testing procedure is done for several generations. For this purpose first the PHYRE Framework and acquiring of the data is described. After that the score function is explained and finally the implementation of the approach is described in detail.

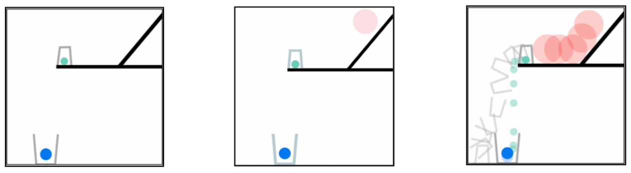

(a) Initial scene (b) Placing an action (c) Execution of action

Fig. 1: Example of the PHYRE simulations. Image a) shows the initial scene. Image b) shows the initial scene with a placed action. Image c) shows the execution of the action placed in the initial scene

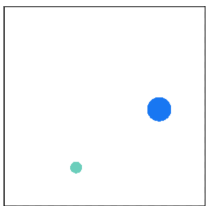
A. Dataset

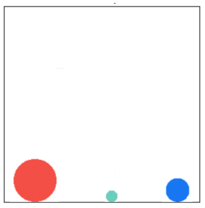
The used data for this approach is provided by the PHYRE framework, which consists of different 2D puzzles that are obeying the rules of Newtonian physics. The scene of a puzzle can contain seven different object types. Each of those object types corresponds to a different color. Black is for the static scene object, and gray for the dynamic scene object, green and blue are used for dynamic objects that are part of the goal state and finally red is the placed action object. The goal of the puzzle is the touching of the green and blue object for at least 3 seconds as consequence of the execution of an action. If no action object is placed, the simulation of the scene will always end in a state that doesn't solve the puzzle.

An example can be observed in figure 1. Image a) shows the initial scene containing black static obstacles, as well as grey dynamic obstacles. The green and blue balls are the objects required for the goal state. Image b) shows the same scene, but now an action object is placed in the top right corner. The placement and the radius of the red ball can vary depending on the chosen action. And Image c) shows the simulation of the placed action in the scene.

There are over 50 different templates, which are differently arranged 2D puzzles. Each template has also 25 different variations, which are tasks. For this approach one task is used, which can be observed in figure 2 image a).

The scene of the chosen task contains a green and blue ball as part of the goal state and no additional scene objects, because this makes it easier to get insight into the physical behaviour of the execution of the scene.
For the acquisition of the data to train the neural networks, actions are randomly sampled from the valid action space of the depicted task in figure 2. An action denotes the placement of a red ball inside the scene with two coordinates and a radius. The executions of the sampled actions are then simulated and the simulation result gives information, if the puzzle is solved by the applied action or not. In the beginning the label of the simulation results are binary: solved and not solved. As in figure 3 depicted the distribution of the labels are heavily biased. Out of 1000 randomly sampled actions, 91.3% of the actions would lead to the puzzle not being solved when simulated. In contrast just 8.7% of the


(a) Initial scene


(b) Final state after execution of an action - not solved

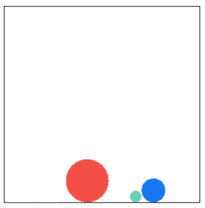

(c) Final state after execution of an action - solved

Fig. 2: Example of the PHYRE simulations. Image a) shows the initial scene. Image b) shows the initial scene with a placed action that doesn't have any impact on the scene. Image c) shows the execution of the action placed in the initial scene that leads to a solved puzzle

actions are solving the puzzles. Because of that the training set needs to be balanced. This is done by copying actions that are solving the puzzle, until a roughly 50/50 ratio of actions that are able to solve the puzzles to actions that don't solve the puzzles are contained in the training set.

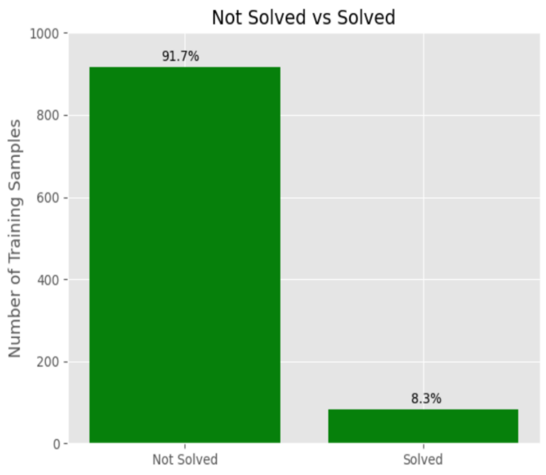


Fig. 3: Binary label distribution

B. Score Function

The solved status is giving implicit information about the relationship between the objects, which is necessary for the scene to be labelled as solved like a distance of zero between the green and blue ball and that this distance has to last for at least three seconds. On the other hand the action space and therefore the arrangement between the objects has a lot more variability for actions that don't solve the puzzle, as indicated by figure 3.
For instance the simulation of a not solved puzzle can end in very different states. Figure 2 shows two of those states. In image a) the red ball doesn't even interact with the other two balls. This means applying this action leads to the same outcome of the simulation as not placing an action at all. Image b) on the other hand shows the simulation for an action that leads to an outcome that lead to the green and blue ball touching.
Three different metrics were used in order to calculate the score: distance, contact and time. Those three metrics are mirroring the requirements of the objective. The distance between the blue and green ball needs to be zero and this has to be the case for at least three seconds. And finally the red ball must have an interaction with the green or blue ball in order to make it even possible for the puzzle to be solved.

- Distance: The score for the distance is calculated by an exponential function
$$\text{distance-score} = 100 \cdot e^{\frac{-distance}{2}}$$
This leads to a maximum of 100 points, if the calculated distance is 0. A cutoff score of 0 is chosen, if the distance between the green and blue ball is larger in the final state, compared to the initial state.
- Contact: As contact between the red ball and either the green or blue ball is the main condition in order to solve the puzzle, flat out contact- score = 25 points are given. Furthermore no dynamic metric is needed as, there has just one frame to be observed, where the two balls are touching.
- Time: This is the metric that has a huge impact on distinction between a solved puzzle and an almost solved puzzle, because of elastic collisions. 10 points are given for each consecutively following frame that shows a distance between the green and blue as being zero. This leads to a maximum of time-score = 30 points, as the objective states that the green and blue object has to touch for at least three seconds, one second corresponds for one frame, and not more.

Finally the total score is calculated by the partially obtained scores for each metric:

$$\text{score} = \text{distance-score} + \text{contact-score} + \text{time-score}$$

This score is now applied as label for the corresponding action instead of a binary solved or not solved characteristic.

C. Approach

As already mentioned the general approach can be divided into a training and testing phase. First the training phase will be explained. For this purpose algorithm 1 is showing the pseudo code for the implementation of the training phase.

Algorithm 1 Training

```
training_samples = []
task = generate_task(task1)
negative_counts = 0

for 100.000 steps do
    action = generate_random_action(task1)
    simulation_result = simulate(action)
    score = score_function(simulation_result)

    if simulation_result.status == NOT_SOLVED then
        negative_counts = negative_counts + 1
        training_samples.append(action, score)
    end if

    if simulation_result.status == SOLVED then
        for negative_counts do
            training_samples.append(action, score)
        end for
        negative_counts = 0
    end if

    model1.train(training_samples)
    model2.train(training_samples)
    model3.train(training_samples)
    model4.train(training_samples)

end for
```

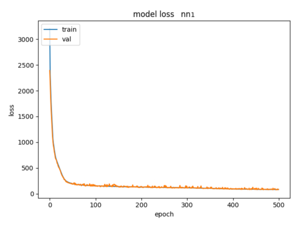
First data for the training of the neural networks needs to be sampled. This is achieved by generating randomly 100.000 valid actions for this specific task shown in figure 2. Each of those actions is then simulated within the simulation environment. The simulation result gives an object with information about the scenes throughout the episode that contains up to 17 frames, one for each second of the simulation. The frames of the simulation results are then used to calculate the score for the action with the help of the score function presented in III-B. In order to assure that the training set is balanced between actions that are able to solve the puzzle and actions that don't solve the puzzle, the negative actions are counted, as those outnumber the positive actions. Each action is appended to a list with all training samples, but the positive action are appended multiple times corresponding to the counter. After that the list of training sample is used to train four neural networks, which are all initialized with different weights.
After the training phase, the testing phase is started. The detailed approach can be observed in algorithm 2. The testing stretches over 10 generations in order to improve the success rate of the chosen actions of the neural networks. During each generation 15000 samples are added to the training set. In order to get those additions to the training set the most inner loop is needed. For each step 1000 test samples are randomly generated. Those test samples are then used as input for each of the trained neural networks. Each of the networks returns a prediction and the mean over those four predictions is then used to calculate the final prediction for the corresponding action. After this is done for all of the 1000 test samples, the action with the highest prediction value out of those 1000 is chosen. This action is now given as input to the simulator. The simulation result is containing the images of the execution. Those images are used in order to determine the actual score that is given for the simulated action, and the action and score are than appended to the training samples. After appending 15000 new action and score pairs to the training samples, all four neural networks are then retrained. After the retraining of the neural networks, a new generation begins.

Algorithm 2 Testing

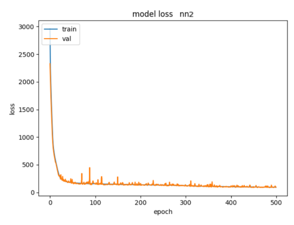
```
task = generate_task(task1)

for 10 generations do
    for 15000 steps do
        testing_samples = []

        for 1000 steps do do
            action = generate_random_action(task)
            prediction1 = model1.predict(action)
            prediction2 = model2.predict(action)
            prediction3 = model3.predict(action)
            prediction4 = model4.predict(action)
            prediction = $\frac{1}{4}\sum_{i=1}^{4} prediction\_i$
            testing_samples.append(action.prediction)
        end for

        best_action = max_score(testing_samples)
        simulation_result = simulate(best_action)
        simulation_score = score_function(simulation_result)
        training_samples.append(best_action,simulation_score)

    end for

    model1.retrain(training_samples)
    model2.retrain(training_samples)
    model3.retrain(training_samples)
    model4.retrain(training_samples)

end for
```

D. Architecture

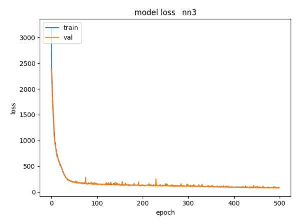
For the architecture of the neural networks the Keras API is used [22]. All four networks are using the same layer structure. They are consisting of four dense layers. The first layer consists of 200 neurons, the second of 100 neurons and the third of 200 neurons. All of the first three layers are using rectified linear unit (ReLU) as activation function. The fourth and last layer has just one neuron and a linear activation function. This is, because the architecture describes a mapping from the x and y coordinates and the radius of an action as input to a single value, the score. As



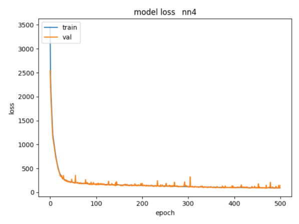
(a) Learning curve of network 1



(b) Learning curve of network 2



(c) Learning curve of network 3



(d) Learning curve of network 4

Fig. 4: Learning curves for the four neural networks trained over 500 epochs

the first four networks are trained, each of the networks is initialized with different weights. The first neural network is initialized with the glorot normal initializer, which draws samples from a truncated normal distribution. The second neural network is initialized with a normal distribution. The third neural network uses an initializer that generates tensors with a uniform distribution and the fourth network is initialized with a glorot uniform initializer. All networks are trained for 500 epochs with a batch size of 32. As optimization algorithm the Adam optimizer with a learning rate of 0.0001 is used and for the error function the mean-squared-error is used. The training loss can be observed in figure 4. Each neural network is trained for 500 epochs. In the beginning the error decreases fast, but it takes a longer time until the error converges close to 0. All loss curves are showing a similar progress, with the validation loss following closely to the training loss, except for some smaller outliers in between.

IV. Evaluation

In this section the results of different experiments concerning the implementation of the described methodology in III is presented.

In order to evaluate the performance of the model, the prediction capability after each generation is tested. For this purpose the 15000 actions that are appended to the train set in each generation are evaluated. The curve for ten generations can be observed in figure 5. The red curve is showing the results for the task depicted in figure 2. In generation 0 the curve starts with 8.3% success rate, before the usage of the neural networks for prediction. This means the results of generation 0 are corresponding to the results of the random agent of the PHYRE framework [18]. In the first generation the networks are used for predicting the best action. Out of those 15000 predictions roughly 61% are actually solving the puzzle. In the second generation and after retraining the neural networks for the first time, the prediction rate goes up by 6%. After that the rate of improvement gets smaller, until after 10 generations the success rate for the predicted action lies around 80%. The exact values for each generation can be found in table I.

TABLE I: Success rate of predicted actions for each generation in percentage

Generation	0	1	2	3	4	5	6	7	8	9	10
Task 1	8.3	61.4	67.3	70.8	72.7	74.9	75.6	76.5	77.3	77.5	77.8
Task 2	7.6	51.7	52.5	54.5	56.1	57.4	58.3	59.4	60.5	61.2	61.5

The blue curve on the other hand shows the results for the model that is trained on task 2 depicted in 1. For this task the curve starts at 7.6% in generation 0. Which is similar to the curve of task 1. In generation 1 and after the first training of the neural networks the value rises to 51%. After that the performance increases about roughly 1% for each generation, up to 60% for generation 10. Those results for task 2 are around 20% worse than for task 1. One reason could be that the score function is better suited for more clear scenes, as more obstacles and clutter are making it harder for the networks to predict actions. Nevertheless in both cases the networks are able to predict suitable actions. However the DQN agent of the PHYRE framework [18] is able to reach over 97% success rate for 100 attempts.

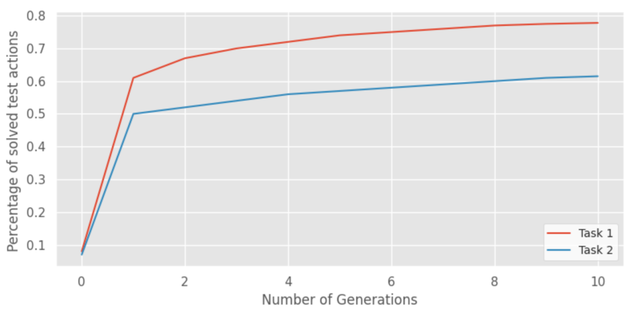


Fig. 5: Percentage of solved puzzles given 1000 actions for two different tasks

As next metric the score function is evaluated. Figure 6 is showing the distribution of the scores for the 100.000 training samples. The scores for the actions that are solving the task are indicated by values over 140. For the scores of the actions that don't solve the task the distribution is broader. Nevertheless over 30.000 actions have a score of zero and therefore don't have any effect on the scene at all. For a score of 25 roughly 10.000 actions having contact with either the green or blue ball, but don't bring the objects closer together. And finally roughly 20.000 actions are resulting in a state that brings the two objects very close to each other.

On the other hand figure 7 shows the distribution of the labels after the prediction for 15.000 test samples. The networks were trained on task 1 and for 10 generations. It can be observed that all values are above 80 with a peak of

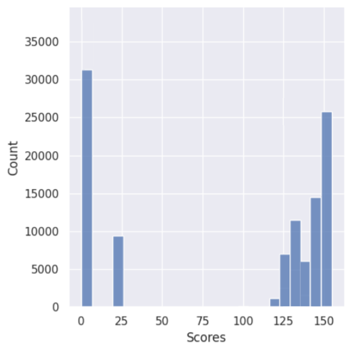


Fig. 6: Percentage of solved puzzles given 1000 actions for two different tasks

5000 actions reaching a value of 150. There are no values close to zero. This indicates that the network is indeed able to differentiate between tasks that don't have any impact on the scene and actions that are close to solving the task or actually are solving the task.

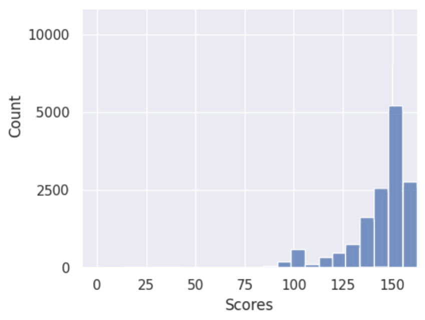


Fig. 7: Percentage of solved puzzles given 1000 actions for two different tasks

V. Conclusion

PHYRE - a benchmark for physical reasoning - is used to train an agent that is able to solve various classical mechanics puzzles in a 2D physical environment. For this purpose this work proposes an agent that is able to pick up on the needed physical reasoning of the given tasks and solve them. The proposed agent consists of four neural networks which are used as action-prediction model, in order to predict the best action for a given task. It is shown that each generation shapes the outcome from a coarser understanding of the scene to a more specific one and improves the prediction capabilities of the network for finding an action that solves the puzzle and therefore being able to differentiate between futile and nonsensical actions. Limitations of this approach are that a lot of samples are needed and thus the retraining throughout generations takes a lot of time and memory space. Furthermore the results differ for different tasks and therefore the score function could be reshaped to fit the needs of scenes that are containing more obstacles.

References

[1] R. Baillargeon, "Physical reasoning in infancy." *Advances in infancy research*, vol. 9, 01 1995.

[2] C. Völter and J. Call, *Causal and inferential reasoning in animals*, 01 2017, pp. 643–671.

[3] S. Hespos, A. Ferry, and L. Rips, "Five-month-old infants have different expectations for solids and liquids," *Psychological science*, vol. 20, pp. 603–11, 05 2009.

[4] N. Neupaertl, F. Tatai, and C. Rothkopf, "Intuitive physical reasoning about objects' masses transfers to a visuomotor decision task consistent with newtonian physics," 02 2020.

[5] M. McCloskey, A. Washburn, and L. Felch, "Intuitive physics: The straight-down belief and its origin," *Journal of experimental psychology. Learning, memory, and cognition*, vol. 9, pp. 636–49, 11 1983.

[6] P. Battaglia, J. Hamrick, and J. Tenenbaum, "Simulation as an engine of physical scene understanding," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, 10 2013.

[7] R. Zhang, J. Wu, C. Zhang, W. Freeman, and J. Tenenbaum, "A comparative evaluation of approximate probabilistic simulation and deep neural networks as accounts of human physical scene understanding," 05 2016.

[8] C. Finn, I. J. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," *CoRR*, vol. abs/1605.07157, 2016. [Online]. Available: http://arxiv.org/abs/1605.07157

[9] W. Li, S. Azimi, A. Leonardis, and M. Fritz, "To fall or not to fall: A visual approach to physical stability prediction," *CoRR*, vol. abs/1604.00066, 2016. [Online]. Available: http://arxiv.org/abs/1604.00066

[10] A. Lerer, S. Gross, and R. Fergus, "Learning physical intuition of block towers by example," *CoRR*, vol. abs/1603.01312, 2016. [Online]. Available: http://arxiv.org/abs/1603.01312

[11] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. B. Girshick, "CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning," *CoRR*, vol. abs/1612.06890, 2016. [Online]. Available: http://arxiv.org/abs/1612.06890

[12] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: visual question answering," *CoRR*, vol. abs/1505.00468, 2015. [Online]. Available: http://arxiv.org/abs/1505.00468

[13] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu, "The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision," *CoRR*, vol. abs/1904.12584, 2019. [Online]. Available: http://arxiv.org/abs/1904.12584

[14] T. Winograd, "Procedures as a representation for data in a computer program for understanding natural language," 10 2004.

[15] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, F. Li, C. L. Zitnick, and R. B. Girshick, "Inferring and executing programs for visual reasoning," *CoRR*, vol. abs/1705.03633, 2017. [Online]. Available: http://arxiv.org/abs/1705.03633

[16] M. Smith, "Running the table: An ai for computer billiards." vol. 1, 01 2006.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[18] A. Bakhtin, L. van der Maaten, J. Johnson, L. Gustafson, and R. B. Girshick, "PHYRE: A new benchmark for physical reasoning," *CoRR*, vol. abs/1908.05656, 2019. [Online]. Available: http://arxiv.org/abs/1908.05656

[19] R. Girdhar, L. Gustafson, A. Adcock, and L. van der Maaten, "Forward prediction for physical reasoning," 2020.

[20] N. F. Rajani, R. Zhang, Y. C. Tan, S. Zheng, J. Weiss, A. Vyas, A. Gupta, C. XIong, R. Socher, and D. Radev, "Esprit: Explaining solutions to physical reasoning tasks," 2020.

[21] A. Harter, A. Melnik, G. Kumar, D. Agarwal, A. Garg, and H. Ritter, "Solving physics puzzles by reasoning about paths," 2020.

[22] Keras Special Interest Group (Keras SIG). (2018) Layers api. [Online]. Available: https://keras.io/api/layers/core_layers/dense/