I. Personal Section - How did I get interested in the field of Machine Learning?

As I began my journey into high school research in my freshman year, I was compelled to find a field of study that combined my passion and background in biology and my proficiency in computer science. Moreover, artificial intelligence (AI) has always been a topic of interest in the back of my mind, and I have to admit that much of my early exposure to AI came from the countless media attention it received. In every YouTube video, TV ad, etc., the possibilities of AI seemed truly endless, so much so that it stirred viral fear over the potentially dire consequences of AI's rapid and uncontrolled growth. However, beyond the public attention, debates, and skepticism, what captivated me the most was the mere concept of 'artificial' intelligence itself. Could we truly replicate the intelligence of our human mind? Much of our brains have yet to be explored, so could we truly create an 'artificial' version of something we don't fully understand? All these questions began to race through my mind, and they ultimately motivated me to learn more about AI and the field of machine learning while also providing me the impetus to discover a newfound passion that perfectly encompassed my then current interests and curiosities: developing AI solutions for medical applications.

My first AI project revolved around creating an Artificial Intelligence solution for the efficient and accurate detection of diabetic retinopathy. Independently, I researched how to train neural network applications, preprocess image data, and experiment with different parameters such as the number of layers, filter size, etc. This project, though perhaps the simplest of those I have done, provided me with the foundation and drive to gain proficiency in developing machine learning applications for future, more difficult problems. The following year as a sophomore, I

conducted a more intricate image processing project on developing an AI solution to segment and classify brain tumor MRI images of glioblastoma patients to improve their treatment planning and their odds of survival. After my work on this project, I had become fully immersed into the world of machine learning (ML) and had developed a greater understanding and sense of confidence in my abilities that then motivated me to explore other avenues in ML beyond image processing.

It was then that I discovered a subfield of ML known as Reinforcement Learning (RL). I came across the field while reading about autonomous navigation, and I immediately became intrigued by the complexity and novelty of the concept behind RL applications. In image processing, an image dataset with ground truth labels (correct answers) enables a relatively simple mode of model training: the model is randomly initialized with model parameters which are sequentially updated based on model performance (does it correctly predict the category of an image or part of an image). In Reinforcement Learning, however, no data exists to train our model; a robot or other agent cannot learn to navigate or perform tasks based solely off of a series of images, making model training exponentially more difficult and variable. Rather, RL tasks follow a trial and error method of learning, similar to how we humans learn a new task. In this manner, the training dataset is actually the past experiences of the robot or agent we aim to train. This extension in difficulty as well as the niche idea of a 'trial and error' training method immediately captured my attention, and my journey in machine learning took on a new direction with an added dimension.

I formally tackled my first RL application in my junior year, during which I worked on engineering medical microbots to navigate the human cardiovascular system and perform critical medical functions, such as blood clot removal and targeted drug delivery to patients. Certainly, it was the most challenging ML project I had done at the time, and it ignited not only a motivation to understand new training algorithms but also a newfound perspective of the growth and importance of machine learning in tackling some of the most complex problems; it was a revelation that ultimately culminated in the project I present in this paper: *Optimally Sparse Deep Reinforcement Learning Solutions for Surgical Robot Task Automation.*

This project was primarily done independently at home on my personal computer, which I used to code and develop the RL algorithms. On a couple of occasions, I also visited the DMC hospital to meet Dr. Ali Abubaker, a premier robotic surgeon in Michigan who taught me the fundamentals of how the dVRK surgical robots operate and even allowed me to watch him conduct a live robotic surgery on a patient. For my project, I certainly had to explore deeper into probability distributions and statistics, which were central to the mathematical innovation behind my project. My advice for high schoolers aspiring to undertake a project combining science and math is to not shy away from seemingly advanced material, as much of it is terminology under which hides familiar ideas and concepts. Moreover, it is critical to not take any shortcuts in model development or testing. Test and account for all possible cases, or as many as you can, since you can never know which tweak or adjustment ultimately leads to a breakthrough. Even failed cases tell volumes about what went wrong and enable you to properly correct errors.

II. Project Section - Project aims, current conclusions, and next steps

As mentioned in the prior section, the title for my project is *Optimally Sparse Deep Reinforcement Learning Policies for Surgical Robot Task Automation.* Though there may be several unfamiliar terms, my goal in this section is to clearly outline each aspect, beginning with the practical aims of the developments I made in this project. In other words, I answer the core questions, “Why automate aspects of robotic surgery and how does my mathematical innovation of introducing sparsity extend universally to all RL applications?”

In today’s world, a robotic surgeon manually manipulates the robot via a console using joystick-like controllers. However, such manual methods of control require the surgeon to spend a significant amount of time performing routine but important tasks, such as suturing or tissue cutting, thousands of times over the course of a single surgery. By automating these tasks, we can improve the accuracy and efficiency of the surgery while also allowing the surgeon more time to focus on the more complex surgical tasks at hand.

As I briefly alluded to in the earlier section, autonomous navigation or performance-related tasks demand the use of Reinforcement Learning because we cannot train an agent based on images and hence we have no training data. Instead, we must train via a trial and error based system, in which the surgical robot learns from its past actions, which play the same role as a traditional dataset. In ‘reinforcement learning', the agent (robot) receives rewards or penalties based on the quality of its actions relative to a defined goal, and after several iterations

the model eventually learns how to take good actions and avoid bad ones to ultimately achieve the target goal.

Yet when I tried to implement traditional Reinforcement Learning policies (the trained model network) for the task of surgical robot task automation, I ran into several challenges. Firstly, the policies did not converge, and I had to utilize advanced techniques and train for significantly long periods of time just to maintain a running model to execute. Moreover, the traditional policies remained 'dense,' meaning they had an excessive number of network connections, or model weights. These so-called 'dense' frameworks require massive amounts of computing power to execute due to their size, and they are also prone to a performance flaw known as overfitting. When a model overfits, it has become over-adapted to the training dataset and therefore performs poorly in real-world execution with different inputs. In theory, this is because a model parameter exists for almost every input-output pair in the training dataset, leading to a model with high variance.

In order to solve these problems of both model convergence and poor model performance as a result of overfitting, I devised a solution plan with two objectives. The first was to introduce 'sparsity' into the RL policies. Sparsity is a biologically inspired concept in which a fewer number of network parameters, as opposed to a larger number, boosts efficiency and robustness in performance. The theoretical explanation for this phenomenon is based on the following intuitive idea: with a fewer number of parameters, there is not a weight for every input-output pair, so the resulting fitted model will be 'smoother' with less variance. Consequently, any new input-output pairs will have a higher likelihood of correlating with end model behavior. The second solution objective is a well-known technique called 'Low-Rank Decomposition.' Using

this technique, the original policy weight matrix is factored or 'decomposed' into three smaller-size matrices that altogether consume less memory than the original. This not only enables efficient model storage but also reduces computational resources and convergence time as well.

To achieve these two objectives, I devised a custom form of what is traditional known as 'L0 regularization.' In conventional L0 regularization, a given neural network is penalized with the number of non-zero weights: the greater the number of non-zero entries in the policy weight matrix, the greater penalty the model receives; the penalty is applied by simply adding the number of non-zero weights to the original reward/loss function (a mathematical formula to assign rewards/penalties to model actions). This incentivizes the policy we train to prioritize not only rewards but also achieve a model with fewer parameters. In other words, we arrive at a 'sparse' counterpart of a dense policy that still delivers high rewards in performance.

The primary issue, however, is that conventional L0 regularization cannot be applied to neural network loss function due to two key problems. Firstly, the graph of L0 penalty, as shown in Figure 1, is non-differentiable at zero, and since adjusting model weights relies on taking derivatives of the loss function with respect to each parameter, traditional L0 prevents such iterative model adjustment (also known as backpropagation). Secondly, because of the binary nature of L0 (every weight is either zero or non-zero), there are 2^theta possible weights matrices to test or consider, which becomes exponentially large and mathematically intractable. To solve both issues, I modified the conventional L0 framework by replacing the count of the number of non-zero elements with probabilities of elements being zero or non-zero. This

removes the binary nature of traditional L0 and solves mathematical intractability, since rather than iterating over all possible 2^theta zero/non-zero matrices, we can simply consider the sum of the probabilities of all elements in the policy matrix being non-zero and add it as a penalty term to the original loss function. Moreover, since the probability distributions are smooth and continuous, the loss function is now differentiable at all points, enabling backpropagation/model optimization to achieve an optimally sparse policy.

Using this mathematical idea to enable the use of L0 on RL policies, I successfully developed a well-performing sparse policy not just for task automation of surgical robots but for a variety of other test applications as well for proof-of-concept purposes. Since my sparsity framework can be universally applied to any RL policy, it was imperative that I illustrate its success in more than one environment. I began benchmarking my sparse L0 framework in three moderately complex environments; these were openly available OpenAI Gym environments for benchmarking RL policies. I found that the sparse policies performed as good or better than the dense policies in every environment and yielded significant improvements in convergence time especially in more complex environments, just as I had hypothesized. I then moved on to a much more complex environment which I am sure every reader is familiar with: SuperMario Brothers. Though many know it as the old-fashioned video game, it is actually a highly complex RL training environment with over 4.3 million parameters. In this more complex environment, my custom L0 framework achieved 93% sparsification (model size reduced to 7 percent of its original) while outperforming the traditional policy by almost two hundred points in rewards! Then finally in the Surgical Robotic Learning Environment (SurROL) hosted by dVRK, the manufacturing company of the surgical robot machines, the sparse policy again significantly

outperformed its dense counterpart, obtaining 36 percent sparsity with higher rewards. Moreover, it was able to achieve 46% decomposition from 'Low Rank Decomposition' without any decay in rewards, whereas the dense policy was able to achieve no degree of decomposition at all without significantly compromising model performance. This pattern was, in fact, observed across all the benchmarking environments: the sparse policy was consistently able to achieve a greater degree of decomposition without decay in rewards over its dense counterpart. Ultimately, these results illustrate efficiency to maximum degree, as not only are the sparse policies significantly leaner in model size and able to be compressed to a greater degree via decomposition, they are also higher performing as well. The results for all the aforementioned experiments are given below in Figure 2.

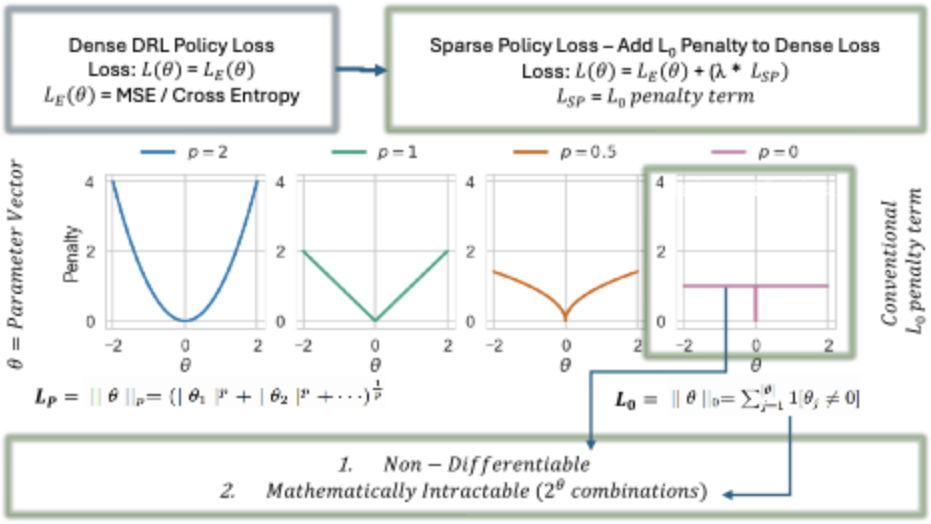
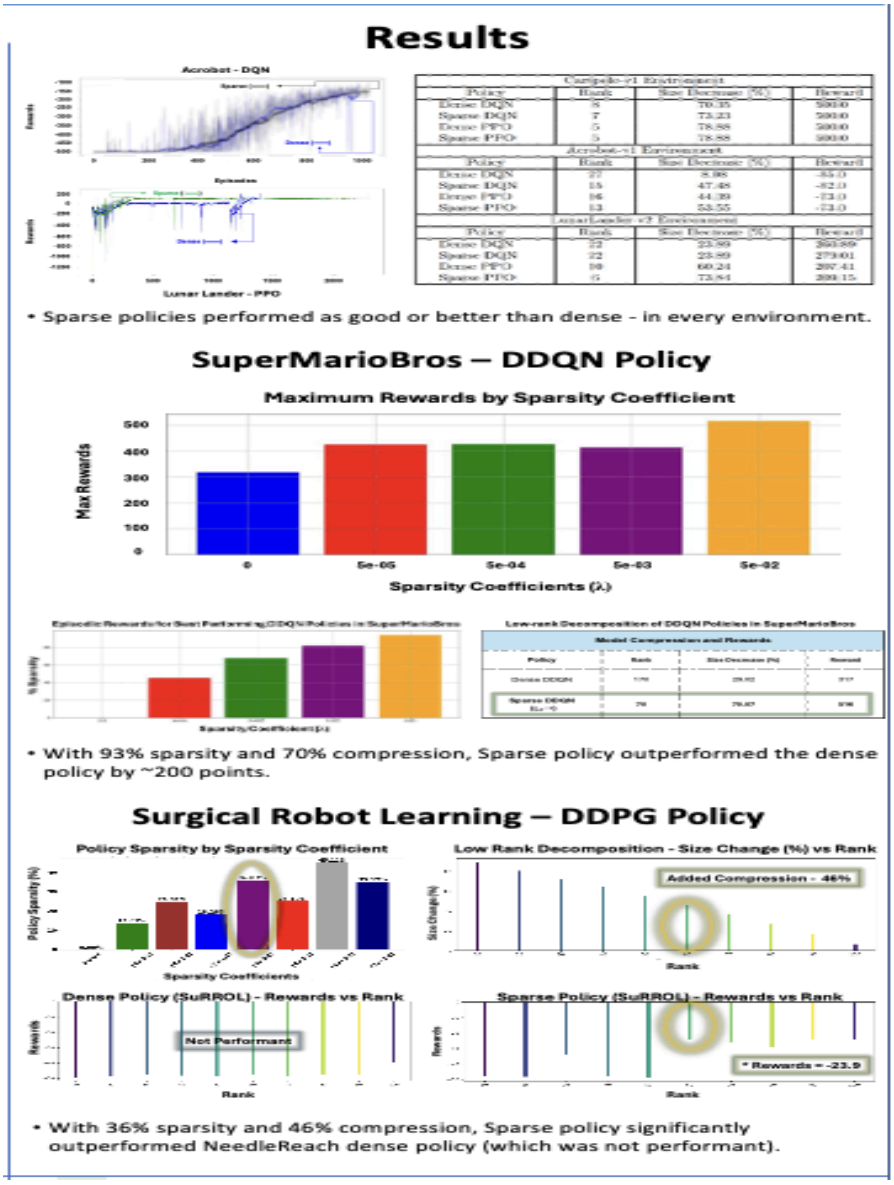


Figure 1: L0, L1, and L2 norm regularization graphs



Low-rank Decomposition of DDQN Policies in SuperMarioBros

Model Compression and Rewards			
Policy	**Rank**	**Size Decrease (%)**	**Reward**
Dense DDQN	170	29.02	317
Sparse DDQN ($5e^{-2}$)	**70**	**70.07**	**516**

Figure 2: Experimental Results:

The top graphs are results from the OpenAI Gym environments. The y-axis represents rewards (how well the model is performing) and the x-axis represents episodes (a unit of training time). The figure illustrates that sparse policies reached their max rewards faster than dense policies and were as good or better than the dense policies in each environment. For the figures in the SuperMarioBros section, the bar graph illustrates the results of different sparsity penalty coefficients (how much to penalize non-zero weights during model training). A coefficient of 0.05 was optimal and outperformed the dense policy by 200 points in rewards. For the final target environment on the bottom, Surgical Robot Learning, the sparse policy again achieved significantly better results: Higher rewards (less negative as indicated by the smaller bar length), and 46% reduction in model size, illustrating how the sparse framework can maximize efficiency by both generating higher performing models that are also leaner and smaller in size.