Learning to Control from Vision: Evaluating Imitation and Supervised Methods in a Competitive Simulated Environment

Jebaraj Vasudevan[1*], Ankita Bagaria[2]

University of Texas at Austin, USA[1], University of Minnesota, Twin Cities, USA[2].

*Corresponding Email: jvasudev@utexas.edu

Abstract

This paper investigates the performance of two learning paradigms—DAgger-based imitation learning and supervised learning—for vision-based control in an autonomous agent setting. We evaluate both approaches in a simulated competitive environment, where agents must interpret visual input and make real-time decisions to achieve task-specific goals. Our experiments show that while both methods are viable, the supervised learning approach consistently outperformed the imitation-based method under time constraints. The final vision-based controller achieved low-latency inference (<100ms per frame on CPU) and demonstrated superior game performance across all difficulty levels. These findings offer valuable insights for selecting control strategies in real-time computer vision applications.

More broadly, our findings contribute to the growing body of research at the intersection of deep learning and real-time control, offering practical guidance for designing robust, efficient agents in fast-paced or safety-critical environments. The methods and insights presented here can inform future development in domains ranging from robotic automation to augmented reality, where rapid perception-action loops are essential.

Keywords: Computer Vision, Convolutional Neural Network, Reinforcement Learning, Vision-Based Control, DAgger (Data Aggregation), Imitation Learning

1. INTRODUCTION

In this paper, we evaluate and compare the effectiveness of two distinct learning paradigm - reinforcement learning with imitation (specifically, a DAgger-based approach) and supervised learning - in the context of a vision-driven control task. To conduct this study, we implemented and tested both methods in a simulated interactive environment, using a competitive setting inspired by a dynamic computer vision task.

Our objective was to develop autonomous agents capable of perceiving visual input and making control decisions to achieve a specified goal within a constrained timeframe. While both approaches were viable, a thorough performance comparison led us to adopt the vision-based supervised learning strategy due to its superior accuracy, stability, and computational efficiency under time-sensitive conditions.

The resulting model demonstrated strong real-time performance, with inference latency below 100 ms per frame on CPU hardware, and consistently outperformed baseline AI agents across varied difficulty levels. This work highlights key trade-offs between reinforcement-based imitation and direct supervision in vision-based control tasks, offering insights applicable to a wide range of real-world autonomous systems beyond the chosen benchmark environment.

2. LITERATURE SURVEY

In recent years, vision-based control has emerged as a powerful paradigm for real-time autonomous decision-making. End-to-end approaches train convolutional neural networks (CNNs) to map raw pixels directly to control commands. Levine et al. [1] demonstrated that deep visuomotor policies can be learned end-to-end on robotic platforms, achieving complex manipulation tasks without handcrafted

features. Similarly, Bojarski et al. [2] showed how a single CNN trained on human driving data can pilot a vehicle in real time, underscoring the viability of supervised vision-to-control pipelines in safety-critical domains.

Imitation learning techniques aim to learn policies by mimicking expert behavior. Ross et al. [3] introduced DAgger (Dataset Aggregation), an iterative no-regret algorithm that collects corrective labels from the expert on states visited by the learned policy, significantly reducing compounding errors compared to naïve behavioral cloning. While DAgger has proven its theoretical appeal, its reliance on expert queries can become a bottleneck in high-frequency control settings.

An alternative to full end-to-end models is a two-stage pipeline combining perception and control. U-Net and its variants have become a standard for dense prediction tasks, segmenting objects of interest with high precision and efficiency. Ronneberger et al. [4] originally applied U-Net to biomedical images, but subsequent work has repurposed its encoder–decoder structure for real-time object detection in robotics, enabling modular controllers to act on detected object locations rather than raw pixel grids.

The paper by Vasudevan [7] shows the difference between the several attention variants on a Question Answering model on the SQuAD data.

Deep reinforcement learning (DRL) methods tackle vision-based control by learning from trial and error. The landmark Deep Q-Network (DQN) by Mnih et al. learned to play Atari games directly from pixel inputs, illustrating that purely reinforcement-based agents can achieve human-level performance in complex visual tasks. More recently, Hafner et al. leveraged learned world models to plan in latent space, improving sample efficiency and stability in visually rich environments.

This paper adds to existing literature by benchmarking a DAgger-based learner against a supervised U-Net controller in a competitive, time-critical environment, demonstrating that the supervised approach yields higher accuracy and stability under 100ms inference requirement.

3. PROPOSED SYSTEM

We decided to try two approaches to implement our agents: a Dagger-based imitation controller and a vision-based controller. In the imitation approach, we collected 4800 images and the corresponding action pairs of the AI set to difficulty level 2 which is the highest difficulty level. A CNN model with MSELoss **Error! Reference source not found.** and BCEwithLogitsLoss Eq. (2), were used to train this classification network.

$$\text{MSE} = \frac{\sum_{i}^{n}(y_i - \hat{y_i})^2}{n} \quad (1)$$

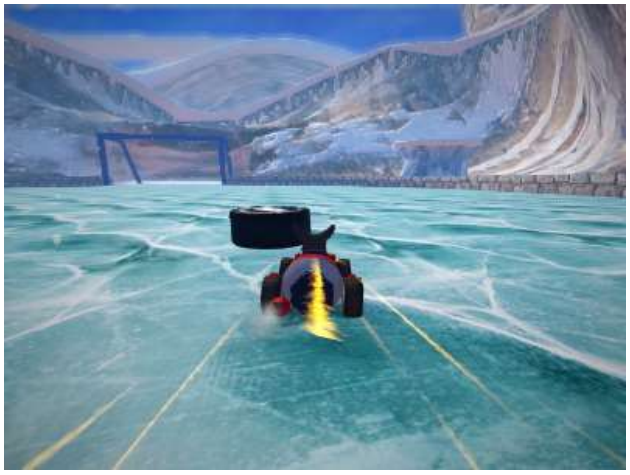
$$BCEWithLogitsLoss(x, y) = max(x, 0) - x.y + log(1 + e^{-|x|}) \quad (2)$$

MSELoss was used for continuous outputs like steer and acceleration while BCEwithLogitsLoss was used for other discrete actions. While imitation learning was quite easy to implement, the Dagger part where we need the AI actions to supervise the agent seemed difficult and time-consuming to implement. In the end, we decided on a vision-based controller approach where we detect the puck and steer the agent accordingly. Specifically, we trained a U-net based segmentation network to detect the puck on the image, compute the angle between the kart and the puck, rotate by that angle and go to the puck, then rotate again and push the puck towards the goal. The data collection and methodology is explained in the sections below.

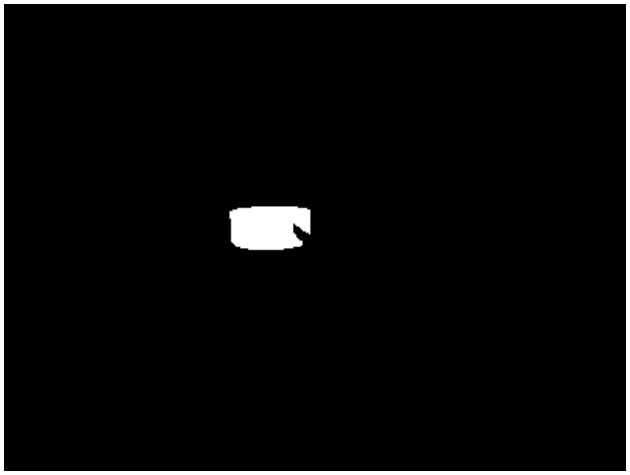
3.1 Data Collection

We generated 12,000 frames from each of the 4 AI-players in the SuperTuxKart game each with

different form the Karts. The total size of the dataset was 48,000 images, along with the corresponding masks, true puck location, and actions taken. The true puck location being a 3D world coordinate was transformed to the 2D normalized image coordinate using the to-image method. This training dataset was used to train the detector network that detects the puck location on the frames and then we used the controller to reorient the player towards the puck.



(a)



(b)

Figure 1: Gameplay: (a) Real world image of Kart and puck (b) Masked image of the puck.

3.2 Network Architecture

The overall network architecture is shown in Figure 2. We implemented the network to detect and locate the puck location on each frame and change the player's direction towards the puck. This network outputs the predicted heatmap of the puck along with the predicted image co-ordinate of the puck's location in the image. The network followed a Fully Convolutional Network architecture closely resembling the general U-net architecture, with a down-sampling section that doubles the number of channels followed by an up-sampling section that halves the number of channels. We started with Initial

state shown in Figure 4 in Appendix that takes in the input image and applies two 3x3 convolutions each followed by a Batch normalization, ReLU activation. Then, a down-sampling step is implemented as shown in Figure 5 of the Appendix that is called in three blocks of the encoder to downsample the image down to the most meaningful pixels. Each down-sampling step consists of the repeated application of two 3x3 convolutions each followed by a Batch normalization, ReLU activation, and a 3x3 max pooling operation with stride 2 for down-sampling. Input normalization is applied lazily through an initial Batch normalization layer. Then, an up-sampling step is implemented as shown in Figure 6 of the Appendix that is called in three blocks of the decoder to upsample the image back to the input size. Each up-sampling section consists of a transpose convolution with a stride of 2 and appropriate padding to half the number of feature channels followed by a Batch normalization and a ReLU activation. Also, each up-sample is connected to the corresponding down-sample using a residual connection to permit easy backpropagation of gradients through the network. Finally, the final layer as shown in Figure 7 of the Appendix that uses 1x1 convolution to output 2 channels one of which is used to generate a heatmap of the puck while the other is spatially argmaxed to get the 2D normalized image location of the puck.

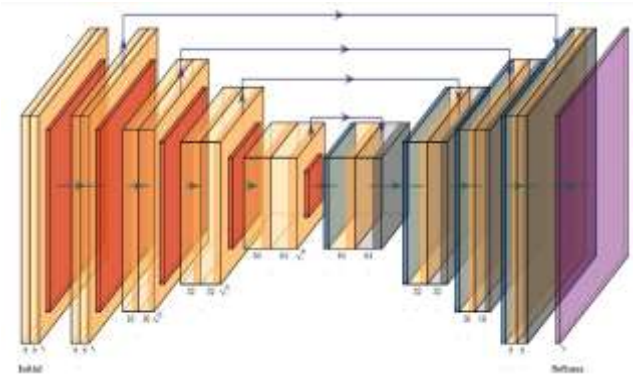


Figure 2: Network Architecture Diagram.

3.3 Network Training

We chose as optimizer Adam Optimizer with learning rate of 0.005 and weight decay of 1e-5 to train the network, along with a learning rate scheduler ReduceLROnPlateau to reduce the learning rate when the loss plateaus and implemented early stopping when the loss no longer decreases. As we have a continuous output suited for regression, we chose SmoothL1Loss Eq. (3), for the location loss, and chose Focal Loss Eq. (4), for the mask loss, and then back propagated on the weighted sum of these two losses.

$$SmoothL1(x,y) = \begin{cases} 0.5.\frac{z^2}{\beta} \ \ if\ |z| < \beta \\ |z| - 0.5.\beta \ otherwise \end{cases} \tag{3}$$

$$Focal\ Loss\ (p,y) = \ -\alpha.(1-p)^{\gamma}.y.\log(p) - (1-\alpha).p^{\gamma}.(1-y).\log(1-p) \tag{4}$$

This allowed faster training of the network in n-epochs=50. The new weights of the network at each epoch are saved only when the loss decreases. The trained network performed exceptionally well in locating the puck as shown in Figure 7 in Appendix. The blue and red circles correspond to the ground truth puck location and the model predicted puck location.

3.4 Controller

In order to convert from 3D kart coordinates to 2D coordinates we omitted the Y-direction since it gives only the vertical coordinate of the puck while the X-Z plane defines the playing field. We used the center of the goal co-ordinates as global constants and chose the "wilbert" cart, because from the image perspective of the player, the model doesn't get confused between the puck and the kart as it has a similar darker color of the puck.

At the start of the game, we initialize status variables which keep track of each function call to the agent, track of the puck location change, and the rescue variables which help rescue the cart when it gets stuck (e.g. stuck to the field's wall). We get the kart's position and velocity from player-info.kart, and use the kart-front vector that defines the orientation of the kart. To locate the puck, we predict the heat map of the puck in the image, the predicted location of the puck in the image, and the list of the peaks of the heatmap of the image along with its scores.

3.4.1. Strategy to check for puck visibility

In order to know if the puck is visible in the image, we filter the list of [score, pixel-x, pixel-y] based on the highest score, so as to filter out the false positives detected by the model and only focus on highest confidence detections. Then, we calculate the mean of these scores, compare it with a threshold and also calculate the sum of the thresholded heatmap of the puck and compare it with the threshold to determine puck visibility. We use both the scores and the sum of the thresholded heatmap to ensure that we avoid false positive detections in the scores while considering only strong signals in the heatmap to detect the presence of the puck.

3.4.2. Strategy to estimate puck size

In order to estimate the size of the puck in the image, we threshold the predicted heatmap-mask to sum up the values with higher confidence, then we normalize by applying torch.sum on the heatmap-mask to have a value in [0,1]. This allows us to estimate the visual size of the puck in the image, i.e. how close or away is the puck from the kart.

3.4.3. Strategy to orient the puck towards opposition goal

If the puck is visible in the image, we use the predicted xcoordinate of the normalized puck location to know if it lies to the left or the right side of the opposition goal. If the puck is not visible and we use the last known puck location for a few steps and if the puck is not visible for quite some time, we reverse back to our goal location. Once the puck is in sight, we compute the angle between the kart's direction and the puck using the arctan2 Eq. (5) formula, which is more stable than the arccos around 0 degrees and 180 degrees.

$$\theta = atan2(y, x) \tag{5}$$

position of the kart w.r.t to the goal is calculated in a similar way using the tangent of the angle between the kart orientation vector and the kart to opposition goal vector to determine if the kart is to the left or right of the goal depending on the sign of the arctan2.

If the kart is within a particular angle on either side of the opposition goal, depending on how far the opposition goal is, an importance metric is calculated which gives more weight to the relative position of the puck compared to the opposition goal angle and how close the kart is to the opposition goal. If not, the kart just goes towards the puck and neglects the angle to the opposition goal.

3.4.4. Strategy to dribble the puck

If the puck is visible, we use the puck size to determine acceleration, steer and nitro values. If the puck is closer to the kart, lesser steer and acceleration is applied with no nitro to gently dribble the puck,

while if the puck is farther away, higher steer, with higher acceleration along with nitro is applied to get to the puck as quickly as possible.

3.4.5. Strategy to recover the karts

In order to recover the kart from getting stuck against a wall, two queues were used to keep track of the last few frames of the kart velocity and if the average of the last few frames is below a certain threshold, then the recovery routine kicks in. The routine for the next few frames reverses the kart towards our own goal and helps the kart to get out of any obstacles.

3.4.6. Other strategies tried but not implemented

- We tried to implement a routine to always face towards the opposition goal, once the kart is no longer facing the opposition goal. But its drawback was that once we try to re-orient the kart towards the opposite goal, we end up losing sight of the puck
- We also tried to implement a goalkeeper approach initially by having one kart play a defensive role while the other kart plays an offensive role. Though this helped to win or draw against the AI, it resulted in low scoring games as one kart is always tied down closer to our own goal. Since the grading heavily weighs high scores compared to draws, we decided to make both players offensive to increase the chances of our team scoring against the AI and other teams

3.4.7. Limitations of our controller

Though our controller was consistently winning against the AI in all levels both as team 0 and as team 1, it has some limitations:

1) Specifically, once the controller is out of phase with the opposite goal, it no longer considers the opposition goal position and may end up scoring against our own goal instead of the opposite goal
2) Also, when both controllers in our team lose sight of the puck, they both fall back to our own goal and keep waiting for the puck to appear in their sight, which sometimes does not work if the AI player also stops playing actively

The controller performed differently in different platforms due to variation though it was able to consistently win or draw against the AI in almost all games.

Figure 3: Supervised Learning Based Agents.

Figure 4: Reinforcement Learning Based Agents - Dagger.

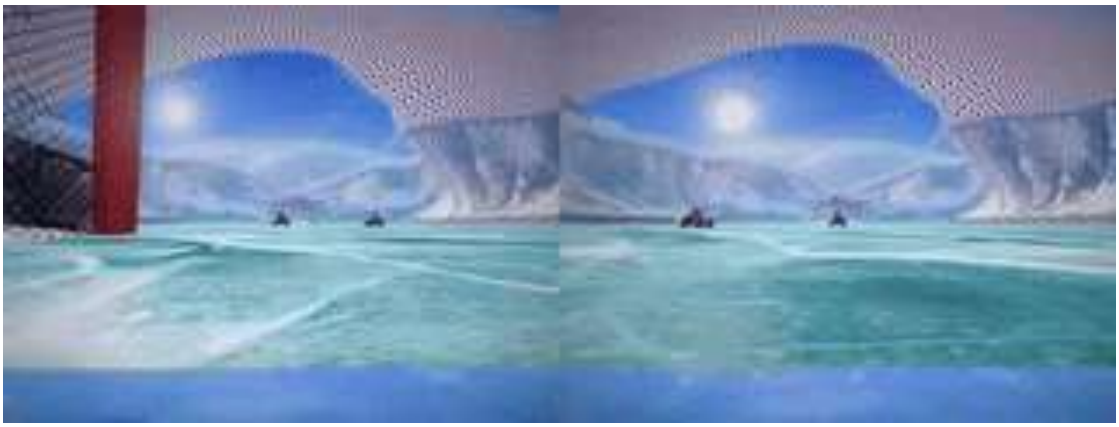


Figure 5: Initial Block.

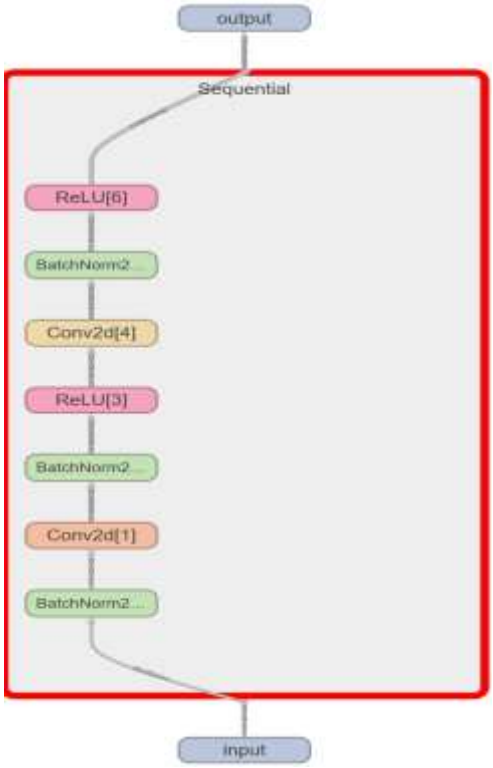


Figure 6: Downstream Block.

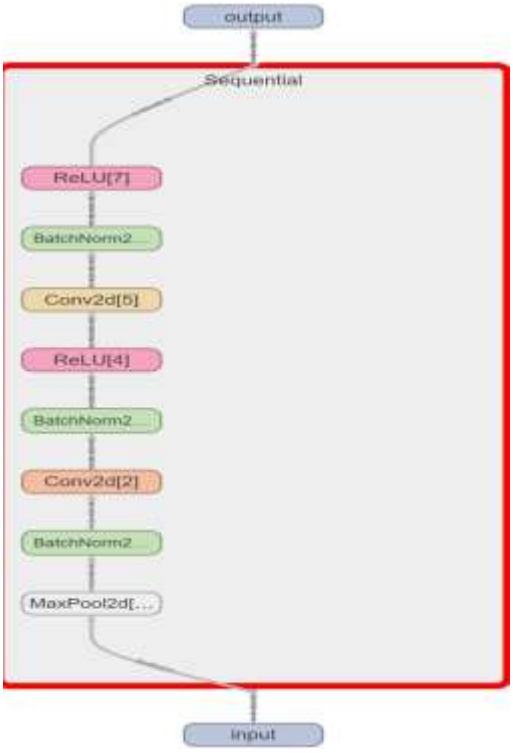


Figure 7: Upsample Block.

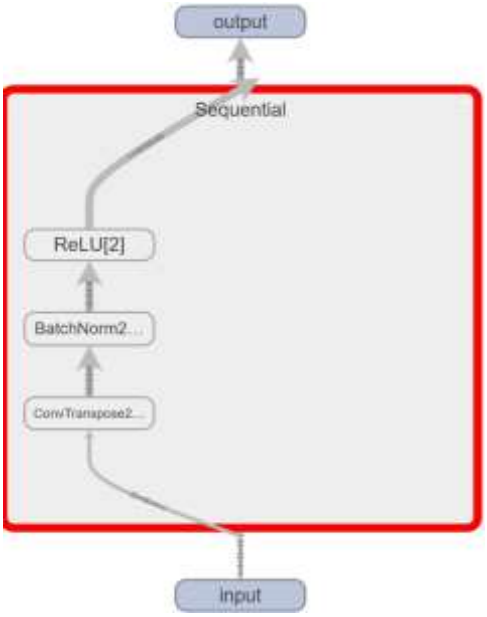
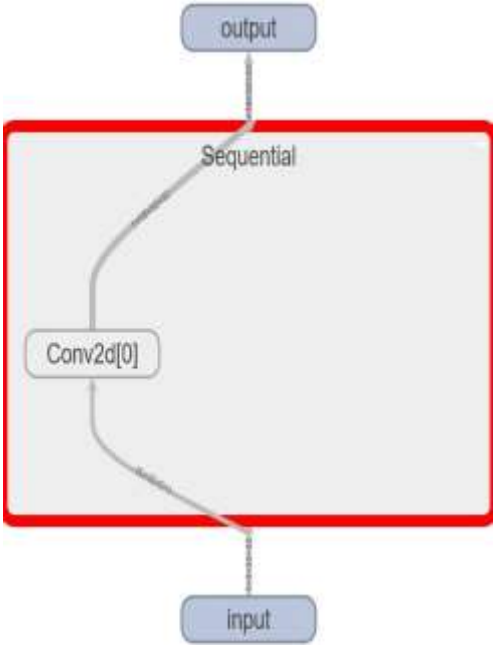


Figure 8: Downstream Block.



4 CONCLUSION

Our vision-based controller strategy of following the puck performed remarkably well vs the Game AI. This strategy of continuously following the puck and keeping it moving towards the opposition goal ensured that our agents always had the puck under their control. We also faced implementation challenges for the vision-based controller approach like creating heuristic rules to score goals. We had to account for multiple scenarios including when the kart gets stuck to the field walls and needs to be rescued, or when the puck is not being seen by both the agents. We tried to overcome these challenges but enumerating infinite scenarios as a strategy seems to be very difficult, so we tried to come up with

a strategy that is generalist enough to ensure the controller performs generally well. For future work, we could try other reinforcement learning methods that use the tuned controller as an oracle with reward defined as the number of goals scored. This approach would help to eliminate the manual fine tuning required for the vision-based controller to work.

REFERENCES

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[2] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Müller, S. Zhang, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *arXiv preprint arXiv:1604.07316*, 2016.

[3] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Fort Lauderdale, FL, USA, pp. 627–635, 2011.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, LNCS, vol. 9351, pp. 234–241, 2015.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Mastering Atari with discrete world models," in *International Conference on Learning Representations (ICLR)*, 2021.

[7] Jebaraj Vasudevan. "Modification and Extension of a Neural Question Answering System with Attention and Feature Variants.", *International Journal of Innovative Science and Research Technology (IJISRT)*, 10(3), 199-208, March 2025. https://doi.org/10.38124/ijisrt/25mar009.

[8] Jebaraj Vasudevan. "Comparative Analysis of Gradient Boosting and Transformer Based Models for Binary Classification of Tabular Data.", *International Journal of Innovative Science and Research Technology (IJISRT)*, 10(3), 466-47. https://doi.org/10.38124/ijisrt/25mar416.