Counter-Strike Retake with Large-Scale Behavioral Cloning

Yifeng Cao
Department of Electronic Engineering
Columbia University
New York, 10025, USA
yc4317@columbia.edu

***Abstract*—This paper presents a novel approach to training an agent for the Counter-Strike: Global Offensive (CS: GO) retake mode using large-scale behavioral cloning. The proposed method leverages a transformer-based architecture and a dataset of human mouse-keyboard demonstrations for training. The dataset includes not only player actions but also round status information, providing valuable data for offline reinforcement learning. The authors address the challenges of interfacing with the game by using RAM hacking for precise information extraction. The agent's observation space is defined by downsampling and cropping the game screen, while the action space is restricted to essential actions for gameplay. Experimental results demonstrate the effectiveness of the proposed method, with different model configurations achieving satisfactory performance. The contributions of this work include the introduction of a sampled human demonstration dataset for the CS:GO retake mode and a transformer-based architecture for sequential decision making in the game. This research represents a significant step towards tackling complex game modes and advancing AI capabilities in competitive gaming**

Keywords—Behavioral Cloning, Transformer

I. Introduction

Deep Learning has proven to be a powerful tool for playing video games, from simple 1970s Atari games to modern competitive Moba games, e.g., DOTA2 [1] and StarCraft2 [3]. Yet most of these works focus on reinforcement learning that requires a standalone environment with a fine-grained API, which requires a huge amount of computational resources to parallel multiple training environments. In openai-five, it took a scaled-up version of Proximal Policy Optimization running on 256 GPUs and 128,000 CPU cores to achieve satisfying results, which makes the following-up work on other games look quite prohibitive. We propose an old-school yet efficient training pipeline with a transformer-based architecture that trains and runs on a single consumer GPU.

Counter-Strike: Global Offensive (CS: GO) is a famous competitive first-person shooter that has the most players to the day 2023, and there were some pioneering works like Tim Pearce et al.[2]. The game has a demanding hardware requirement; thus, it is unachievable to implement RL algorithms directly for a human-like performance. Inspired by previous work[2], we decided to move on to the retake mode that features long sequence modeling and more complex gaming.

TABLE I. THE COMPARASION BETWEEN MODES

Game mode	*Short-term control & Reactive*	*Medium-term Navigation*	*Long-term Strategy & cooperation*
Death match	✓	✓	--
Retake	✓	✓	✓

This paper makes several contributions: 1) Introduces the sampled human mouse-keyboard demonstration datasets for CSGO retake mode to the AI community. 2) Demonstrate a new transformer-based architecture that utilizes spatial information required for more accurate position sensation and better sequential handling of image series. 3) The first major step to conquer competitive game mode that has more complex behavior.

II. Background

A. CSGO Environment

In CS: GO retake mode, players are divided into CTs and Ts, where Ts is spawned with a C4 in the bombsite, and CTs are spawned some distance away. The CT is required to defuse the bomb planted by the T side before the C4 explosion, while the T has to plant and defend the bomb from being defused. Compared with the team deathmatch mode, the retake requires more team collaboration, like covering flashbang and peeking, and it requires a different mindset to play as the C4 counts down. Fights are primarily going on in and around the bombsite, and there is a limited number of pre-defined weapon sets for the player to choose during the freeze period. Most importantly, the grenades are available in the retake mode, allowing for complex tactics and cooperation. In the short term, an agent must control its aim and movement, reacting to enemies. Over the medium term the agent must navigate through map regions, manage its ammunition, and react to its health level. In the long term an agent should manage strategies, adapt to opponents' strengths and weaknesses, and cooperate with teammates.

B. Behavioral Cloning

Behavioral cloning reduces learning from a sequential decision-making process to a supervised learning task. This can be a highly efficient method for learning since an agent is told exactly how to behave, removing the challenge of exploration –

in reward-based learning, an agent experiments to learn strategies by itself.

$$\theta = \arg\min_{\theta} \sum_{i} L\big(a_i, P(\hat{a}|\theta)\big)$$

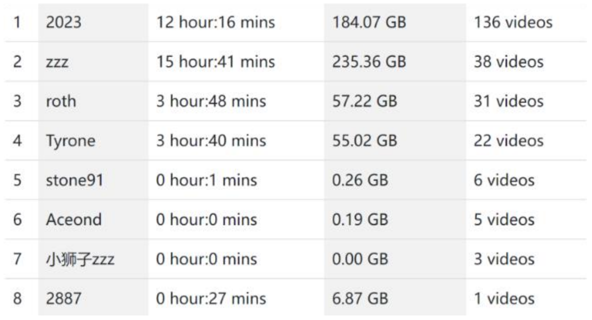
One drawback is that the learned policy can only perform as well as the demonstrator (and in practice, may be worse since it is only an approximation of it). A second is that often only a small portion of the state space will have been visited in the demonstration dataset, but due to compounding errors, the agent may find itself far outside of this.

III. Methods and Data

We put our effort into both the dataset and the architecture. However, the CS: GO doesn't provide an out-of-box method to capture the game info; we handled the interfacing by RAM hacking for the precise information we wanted.

A. *Data*

Firstly, we created a competitive dataset sampled by real volunteers that covers not only the directly sampled player action but also the round status for further work, like the offlineRL approach. The dataset is sampled by our released sample scripts that is tuned for efficient image capturing, keyboard sampling, and real-time RAM hacking. Each frame is tagged with a unique timestamp, and actions were automatically matched to the nearest image with a minimal time gap during label-matching of our post-processing period. The released script is then distributed to our csgo experts for data generation, and then they are uploaded to Aliyun OSS for storage and archiving. We spend too much time on data collection iteration because data collection by nature, is a time-consuming process. To avoid the high cost of human experts, we decided to collect in the classic map, dust2 only (figure 2).



1	2023	12 hour:16 mins	184.07 GB	136 videos
2	zzz	15 hour:41 mins	235.36 GB	38 videos
3	roth	3 hour:48 mins	57.22 GB	31 videos
4	Tyrone	3 hour:40 mins	55.02 GB	22 videos
5	stone91	0 hour:1 mins	0.26 GB	6 videos
6	Aceond	0 hour:0 mins	0.19 GB	5 videos
7	小狮子zzz	0 hour:0 mins	0.00 GB	3 videos
8	2887	0 hour:27 mins	6.87 GB	1 videos

Fig. 1. Summary of Our dataset on retake dust2

Fig. 2. The overview of dust2

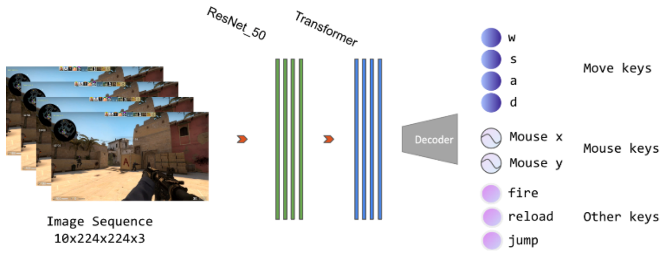


Fig. 3. Model architechture

B. *Model architecture*

As the models, we propose a transformer-based architecture that takes a series of image sequences as input and predicts the following few step actions.

1) Agent observation space

CSGO is typically run at a resolution of around 1920×1080, which is larger than most GPUs can process at a reasonable frame rate, meaning downsampling was required. This gives rise to a tradeoff between resolution, size of neural network, frames-per-second, GPU requirements, and training dataset size. For instance, a lower resolution compromises the agent's skill in longer-range firefights but might allow a deeper neural network to run at more frames per second. Additionally, rather than using the whole screen, a central portion can be cropped, which provides higher resolution for the important region around the crosshair but at the cost of narrowing the field of view.

In this work, the game is run at 1920×1080 resolution and resized to 1024x768, and then the agent crops a central region of 824x498, then downsamples it to 125x200 as a dataset and resized to 224×224 for feature extraction (figure 3). This allows the batched horizon images to be able to train on our single RTX 4090 and extend our agent horizon

Auxiliary Information. The cropped pixel region usefully excludes several visual artifacts that appear in spectator mode but not when actively playing. It also excludes the radar map, score feed, clock, health level, and ammunition. Tim [2] found they were not critical to performance in team deathmatch and excluded them to simplify the design. Since we are considering a different scenario that requires long-term dependency tactical planning and cooperation, we keep the information in our dataset. However, due to insufficient time and manpower for this class project, we leave the multi-vision input for further work that will include the radar, health status, and even round status for offline RL implementation.

2) Agent action space

The action space in CSGO is a mixture of discrete key clicks and continuous mouse movements. To simplify learning, we restrict its output space to actions essential for a reasonable level of play as figure3. Success in firefights requires precise mouse movement (aiming) as well as coordination between actions (e.g., accuracy reduces when moving). This creates two main design challenges: (1) How to model the mouse movement space. (2) How to model actions that are not mutually exclusive (e.g., one might reload, jump and turn left simultaneously). The agent parametrizes mouse movement by changes in x-y coordinates. Whilst previous work has warned us of the deficiency of modeling the mouse movement as a regression task, it may seem natural to treat these as continuous targets, when combined with a mean squared error loss, this led to undesirable behavior in initial experiments (given a choice of two pathways, the agent would output a point midway between the two, which minimized mean squared error!), we still decide to try it out. Our datasets also include continuous mouse data both as pixel and RAM-hacked player angle change.

C. *Interfacing the CS:GO*

A major challenge of the project was solving the engineering task of reliably interacting with the game. CSGO's code base is not open-sourced, and given a widespread cheating problem in CSGO, automated control has been restricted as far as possible.

Image capture: There is no direct access to CSGO's screen buffer, so the game must first be rendered on a machine and then pixel values copied. We used the Win32 API for this purpose; other tested options were unable to operate at the required frame rate. We managed to sample the csgo at 20 frames per second, but the lagging of input is still reported by our data samplers.

Applying actions: Actions sent by many standard Python packages, such as pynput, were not recognized by CSGO. Instead, we used the Windows ctypes library to send key presses and mouse movements and clicks. Recording local actions: The Win32 API was again used to log local key presses and mouse clicks. Mouse movement is more complicated – the game logs and resets the mouse position at high-frequency and irregular intervals. Naively logging the mouse position at the required frame rate fails to reliably determine player orientation. We instead infer mouse movement from our previous hacking CSGOAPI project open-sourced on GitHub.

TABLE II. THE CONFIG OF MODELS

Curve color	***Horizon***	***Batch***	***Model option***
red	20	1	3
green	10	2	3
purple	30	3	3

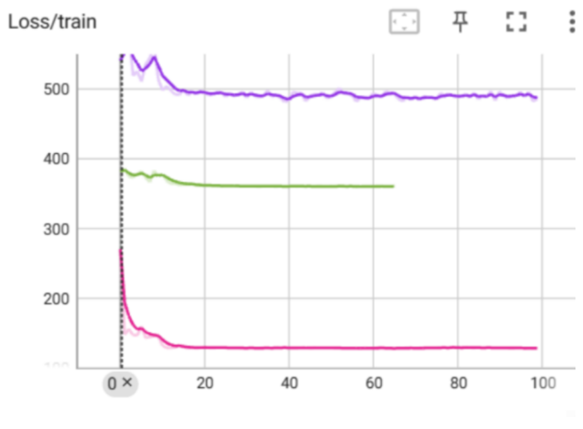


Fig. 4. Loss curves

IV. Experiments

We use three different model settings for the training. In option 1, we freeze the ResNet-50 pre-trained on imageNet, and the features are fed into transformers directly. In option 2, we inserted a trainable fully connector layer in the middle of the pre-trained ResNet-50 and the transformers as a feature projection layer. in option3, we freeze the 3rd and 4th layer in ResNet-50 and keep the rest the same as option 2.

the figure 4 shown 3 trails. we can judge that the loss converges faster with the horizon (the number of previous images fed to the network for action inference) valued at 20 that will be 1 second in real life

We also used our converged model to implement field tests on csgo. The result shows unsatisfying when the agent didn't learn how to move the mouse correctly and kept running into walls. We attribute the behavior to the ResNet architecture

We used because the full resnet features in the latest layer may lose spatial information that is needed for mouse movement and 3d structure sensing. In the following work, we shall implement two alternatives. One is replacing the resnet into Vision Transformers such that the structure information will be kept as large-scale correlations. The other is that we extract only the low-level feature map of ResNet to keep the spatial information, just like what Tim [2] did before

V. Conclusions

We were too ambitious on our project in the first place, so the project was squeezed with emerging new engineering problems like unprecise data or invalid information. It may seems unfinished by viewing, but we know that there is a promising future for our work as we find the correct architecture and training method.

In future work, we think a new dataset is the first thing need further work. Now, the data we have contains many noise (Many experts present bad habits like switching to a knife for no reason). And we know the agent is sensitive to the data, so we think this could be the major reason for failure.

The training server is another thing we need; now we trained all the models on the client-server (one Nvidia 4090); every train costs around one day, which leads to the end that we don't have time to fine-tune.

Besides, our model policy costs nearly 0.1s every step to generate the new step keys output; this frequency can not perform well in the real environment. So, future work needs to focus on narrowing the model structure and improving the running server.

Acknowledgment

As you see, this huge work definitely can not be finished by a single person, so I asked some friends in China to help. Really appreciate Shi Meng(mengshi2022@ia.ac.cn), Weiyu Ma(weiyuma@ia.ac.cn) and my CSGO team. They and all the big fans of CSGO, and they also really want to do something for this game research. Honestly, the dataset cleaning work is finished by Shi Meng. Weiyu Ma helped me to train the model in his Nvidia 4090 and tested the AI agent. At the end, thank to my CSGO game friends, they spent many daily hours for collecting the dataset.

References

[1] Berner, Christopher, et al. "Dota 2 with large scale deep reinforcement learning." arXiv preprint arXiv:1912.06680 (2019).

[2] Pearce, Tim, and Jun Zhu. "Counter-strike deathmatch with large-scale behavioural cloning." 2022 IEEE Conference on Games (CoG). IEEE, 2022.

[3] Vinyals, Oriol, et al. "Starcraft ii: A new challenge for reinforcement learning." arXiv preprint arXiv:1708.04782 (2017)..