

Counter-Strike Deathmatch with Large-Scale Behavioural Cloning

Tim Pearce, Jun Zhu
Tsinghua University

Abstract—This paper describes an AI agent that plays the modern first-person-shooter (FPS) video game ‘Counter-Strike: Global Offensive’ (CSGO) from pixel input. The agent, a deep neural network, matches the performance of a casual human gamer on the deathmatch game mode whilst adopting a human-like play style. Much previous research has focused on games with convenient APIs and low-resolution graphics, allowing them to be run cheaply at scale. This is not the case for CSGO, with system requirements orders of magnitude higher than previously studied FPS games. This limits the quantity of on-policy data that can be generated, precluding pure reward-driven reinforcement learning (RL) algorithms. Our solution uses a two-stage behavioural cloning methodology; 1) Pre-train on a large dataset scraped from human play on public servers (5.5 million frames or 95 hours) where actions are labelled in an automated way. 2) Fine-tune on a small dataset of clean expert demonstrations (190 thousand frames or 3 hours). This scale is an order of magnitude larger than prior work on imitation learning in FPS games, whilst being far more data efficient than pure RL algorithms.

Index Terms—AI, behavioural cloning, reinforcement learning

Video introduction: <https://youtu.be/rnz3lmfSHv0>
Code, model & datasets: <https://github.com/TeaPearce>

I. INTRODUCTION

Deep neural networks have achieved strong performance in a variety of video games; from 1970’s Atari classics, to 1990’s first-person-shooter (FPS) titles Doom and Quake III, and modern real-time-strategy games Dota 2 and Starcraft II [4, 17, 20, 24, 32]. Such systems typically succeed by running deep reinforcement learning (RL) algorithms at a massive scale – for instance, an actor-critic algorithm used 10,000 years of experience to master Dota 2.

Games without convenient APIs, that can’t be run easily at scale, have received less research attention. Without access to mass-scale simulations, one is forced to explore more efficient algorithms. In this paper we take on such a challenge; building an agent for Counter-Strike: Global Offensive (CSGO), with no pre-existing API, and modest compute resources (4×GPUs for training, 1×GPU at test time, and a single game terminal).

Released in 2012, CSGO is one of the world’s most popular games in player numbers and audience viewing figures. The computational requirements of CSGO are an order of magnitude higher than the FPS games previously studied. For instance, while Doom can be run at 7000 frames-per-second on a single CPU [33], CSGO runs at 200 frames-per-second on a modern GPU.

CSGO’s constraints preclude mass-scale on-policy rollouts, and demand an algorithm efficient in both data and compute,

which leads us to consider behavioural cloning. Whilst prior work has applied this to various games, demonstration data is usually limited to what authors provide themselves. Playing repetitive game modes at low resolution means these datasets remain small (one to five hours – section II), producing agents of limited skill-level.

Our work takes advantage of CSGO’s popularity to record data from other people’s play – by joining games as a spectator and scraping screenshots and inferring actions. This allows us to collect a dataset an order of magnitude larger than in previous FPS works, 5.5 million frames or 95 hours. We use a two-stage approach; initially training a deep neural network on this large noisy dataset, then fine-tuning it on smaller clean expert demonstrations. The resulting agent can play the game with a skill-level around that of the medium-difficulty built-in bot (the rules-based AI available as part of CSGO), or equivalently, a casual human FPS gamer.

Whilst RL research often aims to maximise reward, we emphasise that this is not the exclusive objective of this paper – perfect aim can be achieved through simple geometry and extracting enemy locations (hacks and built-in bots exploit this). Rather, we aim to produce an agent that plays in a humanlike fashion, that is fun and challenging to play with and against.

This paper makes several **contributions**: 1) Provides a blueprint for building data and compute efficient agents for modern games. 2) Proposes a two-stage behavioural cloning approach. 3) First major work on a modern FPS game, and largest-scale behavioural cloning effort in this genre. 4) Introduces the CSGO environment to the AI community.

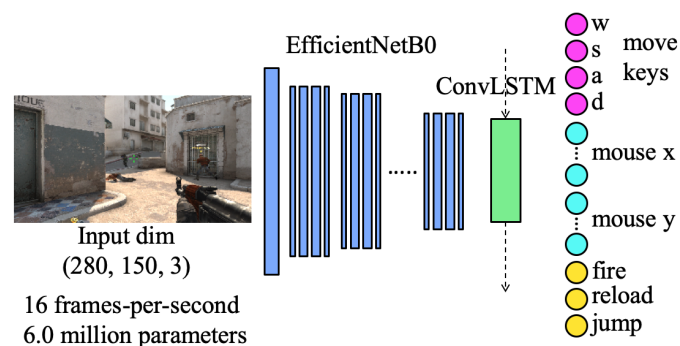


Fig. 1. Overview of the agent’s architecture. Deathmatch mode shown.

II. RELATED WORK

FPS games have proven to be useful environments for RL research [26]. Two 1990’s games have been packaged in convenient APIs. Beattie et al. [3] released DeepMind Lab, built around Quake 3 (originally 1999), and Kempka et al. [19] introduced VizDoom (originally 1993). These environments are basic in comparison to CSGO (originally 2012 and continuously updated), using low resolution textures, a smaller action space, and orders of magnitude less compute – e.g. VizDoom allows simulation at 7000 frames-per-second on a single CPU core [33], whilst CSGO runs at around 200 frames-per-second on a modern GPU.

These FPS environments have attracted much research, many applying standard reward-based learning such as actor-critic methods or Q-learning, e.g. [17, 20]. Several efforts have trialled behavioural cloning, with demonstration datasets of around one hour [9, 13, 18] (table I). There have also been explicit efforts toward building humanlike agents [10, 15]. Our work stands out both as the first to tackle a *modern* FPS, and as the largest-scale effort in behavioural cloning.

Imitation learning has been explored in other genres most often at small scale (1-5 hours – table I), with authors recording demonstration data themselves. There are several notable exceptions; Go (30 million frames) [30], Starcraft II (971,000 replays) [32], and Minecraft [11] (500 hours). Using these larger datasets reasonable performance *could* be achieved (e.g. Vinyals et al.’s agent achieved a rank in the top 16% of human players), and they have inspired much follow up work. We hope the dataset we contribute in the CSGO environment will be valued similarly.

The computational difficulty of generating on-policy data for CSGO makes the blossoming field of offline RL [22] very relevant, where there has been recognition that leveraging existing datasets for tasks typically tackled through pure RL could greatly improve efficiency. Benchmarks and datasets in offline RL are often algorithmically generated [8] – this has found particular favour in Atari games [1]. We hope our large-scale human demonstration dataset might find use in this field.

Relatively little research effort has been applied to Counter-Strike, likely due to there being no API to conveniently interface with the game, and difficulty in mass roll-outs. Relevant machine learning works include predicting enemy player positions using Markov models [16], predicting the winner of match ups [23], and tuning bot hyperparameters using genetic algorithms [7]. It has also been studied from various societal and cultural perspectives, e.g. [12, 27]. Ours is the first work to build an AI from pixels for CSGO.

III. BACKGROUND

This section describes the CSGO environment, and briefly outlines behavioural cloning.

A. CSGO Environment

CSGO is played from a first person perspective, with mechanics and controls that are standard across FPS games – the keyboard is used to move the player

TABLE I
COMPARISON WITH PRIOR WORK USING IMITATION LEARNING IN GAMES.

Citation	Game	FPS?	Dataset size	From pixels?
[13]	In-house game	✓	0.75 hours	✓
[9]	Quake 2	✓	1 hour	✗
[18]	Various incl. Doom	✓	0.75 hours	✓
[6]	Super Mario Smash Bros	✗	5 hours	✓
[14]	Atari	✗	1 hour	✓
[5]	NecroDancer	✗	1.5 hours	✓
[32]	Starcraft II	✗	4,000 hours	✗
[30]	Go	✗	30 million frames	✗
[11]	Minecraft	✗	500 hours	✓
Our work	CSGO	✓	95+3 hrs	✓

left/right/forward/backwards, while mouse movement turns the player horizontally and vertically, serving both to look around and aim weapons. In CSGO’s full ‘**competitive mode**’, two teams of five players win by eliminating the opposing team, or completing an assigned objective. Success requires mastery of behaviour at three time horizons; In the **short term** an agent must control its aim and movement, reacting to enemies. Over the **medium term** the agent must navigate across map regions, manage ammunition and react to its health level. In the **long term** an agent should manage its economy, plan strategies, adapt to opponents’ strengths and weaknesses and cooperate with teammates.

As the first attempt to play CSGO from pixel input, we do not consider the full competitive mode. Instead we focus on two simpler modes, summarised in table II. All CSGO game modes are partially observable, stochastic environments.

‘**Aim training mode**’ provides a controlled environment for players to improve their aim, recoil control and reaction speed. The player stands fixed in the centre of a visually uncluttered map, while unarmed enemies run toward them. Players cannot take damage, and ammunition is unlimited. This constitutes our simplest environment.

‘**Deathmatch mode**’ rewards players for eliminating any enemy on the opposing team (two teams, ‘terrorists’ and ‘counter-terrorists’). After dying a player revives at a random location. Whilst it does not require the long-term strategies of competitive mode, other elements are intact. It is played on the same maps, with the full variety of weapons available. Ammunition must be managed, and the agent should distinguish between teammates and enemies.

We consider three difficulty settings of deathmatch mode, all on ‘dust2’ map, the agent on the terrorist team and ‘AK47’ equipped. 1) **Easy** – built-in bots on easy mode, bots use pistols, reloading not required, 12 vs 12. 2) **Medium** – built-in bots on medium difficulty, any weapon, reloading required, 12 vs 12. 3) **Human** – human players, any weapon, reloading required, 10 vs 10.

B. Behavioural Cloning

In behavioural cloning (a form of ‘imitation learning’) an agent learns to mimic the action, $\mathbf{a} \in \mathcal{A}$, a demonstrator would take given some observed state, $\mathbf{o} \in \mathcal{O}$. Typically the

TABLE II
TIME HORIZON REQUIRED FOR SUCCESS IN EACH GAME MODE.

Game mode	Short-term Reactive & control	Medium-term Navigation & ammo	Long-term Strategy & cooperation
Aim training	✓	✗	✗
Deathmatch	✓	✓	✗
Competitive	✓	✓	✓

agent, parameterised by θ , outputs a probability distribution over possible actions, $\pi_\theta(\hat{\mathbf{a}}|\mathbf{o})$.

Learning is based on a dataset of the demonstrator’s behaviour. For N such pairs, $\mathcal{D} = \{\{\mathbf{o}_1, \mathbf{a}_1\} \dots \{\mathbf{o}_N, \mathbf{a}_N\}\}$. In its vanilla form, behavioural cloning uses some loss function, $l : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ (e.g. cross-entropy or mean squared error), measuring the distance between predicted and demonstrated actions, and a model is trained to optimise,

$$\theta = \operatorname{argmin}_\theta \sum_i^N l(\mathbf{a}_i, \pi_\theta(\hat{\mathbf{a}}_i|\mathbf{o}_i)).$$

(Though either \mathbf{o}_i or π_θ could be modified to use more information than just the current timestep.)

Behavioural cloning reduces learning a sequential decision making process to a supervised learning task. This can be a highly efficient method for learning [2], 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.

One drawback is that the learnt policy can only perform as well as the demonstrator (and in practise 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 [21, 28] – there is a distribution mismatch at test time, $p_{\mathcal{D}}(\mathbf{o}) \neq p_{\pi_\theta}(\mathbf{o})$.

IV. AGENT DESIGN

This section details the major design decisions of the agent.

A. Observation Space

CSGO is typically run at a resolution of 1920×1080 , which is larger than most GPUs can process at a reasonable frame rate. There is a trade-off between resolution, field-of-view, 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.

In this work the game is run at 1024×768 resolution, and the agent crops a central region of 824×498 , then downsamples it to 280×150 . This allows state-of-the-art network architectures to run at 16 frames-per-second on an average gaming GPU.

Auxiliary Information. The cropped pixel region usefully excludes several visual artefacts which appear in spectator mode but not when actively playing. It also excludes the radar map, score feed, clock, health level and ammunition.

We experimented providing some of these in vector form to the network, but found they were not critical to performance, and excluded them to simplify the design. As such, the agent has no direct knowledge about its health or ammo level.

B. Action Space

We simplify the agent’s actions space to those essential for a reasonable level of play as per figure 1. It excludes other actions such as the ‘walk’ key. We faced two main design challenges:

1) How to model the continuous mouse space. The agent parametrises mouse movement by changes in x & y coordinates. If treated as continuous targets and combined with a mean squared error loss, this led to undesirable behaviour (given a choice of two pathways, the agent outputs a point midway between the two) [25]. Discretising the mouse space and framing it as a classification task was more successful.

The discretisation itself required tuning and experimentation – a finer grid allows more precise control but requires more data to train. The agent uses an innovative unevenly discretised grid, finer in the middle, and coarser at the edges – 19 options for mouse $x \in \{-300, -200, \dots, -10, -4, -2, 0, 2, 4, 10, \dots, 200, 300\}$, and 13 options for mouse $y \in \{-50, \dots, -10, -4, -2, 0, 2, 4, 10, \dots, 50\}$. This reflects that it’s more important for a player to be able to make fine adjustments when aiming, compared to when turning large angles, and also that vertical movements tend to be of lower magnitude than horizontal movement. Histograms of mouse movements in human play in figure 2 also justify this choice, since the majority of mouse adjustments are of low magnitude.

2) How to model actions that are mutually inclusive. It’s possible that a player might reload, jump and turn left simultaneously. To allow this, independent losses are used for each action – binary cross entropy losses for keys and clicks, and multinomial cross entropy losses for each mouse axis. As such the agent outputs the *marginal* distribution of each action rather than the *joint* distribution, i.e. it assumes that each action is independent of all others. This simplifying assumption worked adequately well empirically, though in specific situations it may be harmful – for instance if choosing to step left and reload behind cover, or remain static and fire.

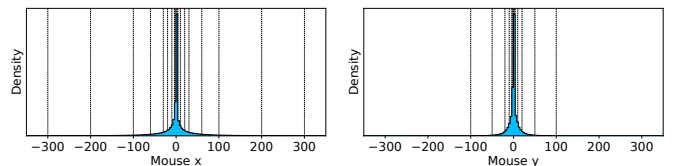


Fig. 2. Mouse movement histograms in the online dataset. Note mouse x has a larger variance than mouse y . Dashed lines overlaid show discrete options output by the agent.

C. Neural Network Architecture

The agent’s architecture is summarised in figure 1. An EfficientNetB0 [31] forms the trunk of the network, initialised with ImageNet weights. We use only the first six residual stages to avoid losing too much spatial information about the input via additional pooling layers. For an input of $280 \times 150 \times 3$, a feature map of dimension $18 \times 10 \times 112$ is output.

To allow the agent to sense motion of itself and others, we employ a convolutional LSTM layer [29]. Initial experiments indicated that a stacked sequence of inputs was a less promising approach. A linear layer connects the output layer, followed by either a softmax (for mouse x , mouse y) or sigmoid (for w , a , s , d , fire, reload, jump).

D. Test Time

The agent parameterises a probability distribution over each action independently. Combinations of actions may be applied at every time step. At test time, each action can either be selected probabilistically $\tilde{\mathbf{a}} \sim \pi_\theta(\hat{\mathbf{a}}|\mathbf{o})$, or according to the highest probability, $\tilde{\mathbf{a}} = \operatorname{argmax}_{\hat{\mathbf{a}}} \pi_\theta(\hat{\mathbf{a}}|\mathbf{o})$.

Selecting movement keys and mouse movement probabilistically produced jerky, unnatural movement, so are selected via argmax . Certain actions – reload, jump, fire – seldom exceed the 0.5 threshold required to be chosen by argmax , so are selected probabilistically.

With actions being applied 16 times a second, mouse movement can appear stilted. This can be artificially increased to 32 by halving the mouse input magnitude and applying twice with a short delay.

V. METHODS & DATA

This section introduces the methodology used for training the agent, describes collection of the demonstration datasets, and summarises some training details.

A. Two-Stage Methodology

One of the difficulties of using behavioural cloning in many applications is that sourcing a large dataset of demonstrations is generally time-consuming and/or costly. Much of the literature in video games manually records demonstrations by using a specially set up machine to log key presses and mouse movements, but this results in small datasets (repetitive game modes in low resolution are no fun!) and systems of limited performance (table I).

Prior work in Starcraft II showed that reasonable performance *can* be achieved through behavioural cloning, provided one has access to a dataset of sufficient size [32]. Whilst Vinyals et al. worked alongside game developer Blizzard, having access to a large dataset of logged states and actions, for many games such access is not possible.

In lieu of such privileges, we developed a two stage method. In stage one, we scrape a large dataset of human play from public online servers. We do not have access to the ground truth actions applied by the player, and instead build an inverse dynamics model to estimate these actions from metadata. This

is used for pre-training a neural network. In stage two, we manually create small clean datasets that the network is fine-tuned on. The clean datasets have several advantages that drastically boost performance.

- Recording gameplay allows clean labelling of the actions.
- We restrict the demonstrator’s action and observation space to match that of the agent.
- There are minor differences in the visuals rendered by the game when viewing players in spectator mode, compared to actively playing, e.g. red damage bar indicators are not displayed in the former.
- The online dataset contains a large variety of play styles and equipment choices. The clean dataset allows the network to specialise to a single high-skill policy.

Combining two datasets in this way allows the agent to learn from the broad state-space coverage in the online dataset, without compromising on the quality of the final policy. The quantity of manual demonstrations required is an order of magnitude smaller than if exclusively trained on.

Note that we have taken steps to anonymise datasets, excluding player handles from the metadata. We have communicated directly with CSGO’s developer Valve about the sharing of the datasets and code from this paper, which has been approved in a research context.

B. Large-Scale Online Demonstrations

This dataset was scraped from official Valve servers by joining in spectator mode, and running a script both to capture screenshots (processed as in section IV) and metadata at 16 frames-per-second. In total the dataset was 680GB / 95 hours / 5,500,000 frames. Note the naming of this dataset as ‘online’ refers to the source being online Valve game servers, and *not* to the offline/online learning paradigms in RL. We defer details around interfacing with the game to the [code repository](#).

The script tracked the current best performing player in the server to collect higher-skill demonstrations. Periods of player immobility were filtered out in post-processing.

Metadata does not contain the actions that were applied by the player. Rather, it contains information about the player state (e.g. weapon selected, available ammunition, health, score), position on map (x, y, z coordinates), velocities, and orientation (roll and yaw). Letting \mathbf{m} represent this metadata, the dataset can be denoted, $\mathcal{D}_{\text{online}} = \{\{\mathbf{o}_1, \mathbf{m}_1\} \dots \{\mathbf{o}_N, \mathbf{m}_N\}\}$.

We developed a rules-based algorithm for the inverse dynamics model, $\bar{\mathbf{a}}_i = f(\mathbf{m}_{i-1}, \mathbf{m}_i, \mathbf{m}_{i+1})$. Whilst some actions were straightforward to infer (e.g. firing is detected if ammunition decreased between two time steps). Others required testing and tuning. For instance, inferring keys moving a player forward/backwards/left/right, is an ill-posed problem – there can be many other reasons for a change in velocity, such as weapon switching (heavy weapons makes players move slowly), bumping into objects, or taking damage. There were also inconsistent time lags between an action’s application, its manifestation in the metadata, and observing the change on screen. See [dm_infer_actions.py](#) for the full algorithm.

We tuned the inverse dynamics model until it was able to infer actions in most scenarios tested. Whilst this required a significant reverse-engineering effort, the value was in its scalability – once written it could scrape gameplay continuously for days at a time, providing a quantity and variety of demonstrations that we couldn’t produce manually.

C. Clean Expert Demonstrations

We created two clean datasets using a terminal set up to precisely log actions and take screenshots, one for the deathmatch game mode and one for the aim train mode. These datasets were 24GB / 3.3 hours / 190,000 frames and 0.8 hours / 6GB / 45,000 frames respectively.

We used a strong human player to provide the data (top 10% of CSGO players, ‘DMG’ rank). This player was only allowed to use actions the agent can output. The game was run at 1024×768 resolution. The audio was muted and radar covered up to mimic the agent’s observation space. For some of the demonstrations in easy and medium mode, we slowed the game to half speed (dropping the capture rate accordingly) to further improve the quality of the demonstrations.

D. Training Details

The agent is initially trained on the online dataset (validating on medium deathmatch mode every two epochs). From this pre-trained checkpoint, fine-tuned versions were created by further training on one of the clean expert datasets (validating on the relevant map and mode every four epochs). A batchsize of 4 and sequence length of 96 frames (6 seconds) were used (LSTM states are reset between each sequence). Data augmentation was applied to image brightness and contrast, but not to spatial transformations, since this would invalidate mouse labels.

In addition to the losses discussed, the agent outputs and optimises a value function estimate ($v_t = r_t + \gamma v_{t+1}$, where, $r_t = 1.0 \text{ kills}_t - 0.5 \text{ deaths}_t - 0.02 \text{ shoot}_t$, for binary indicator variables, $\text{kills}_t, \text{deaths}_t, \text{shoot}_t$). This may have the effect of providing extra supervision as an auxiliary task. In this paper the value function estimate is not used for any further purpose.

Ten different models were trained on the online dataset under various hyperparameter settings. Training for each model used 4× Titan X GPUs – time for one epoch on the online dataset varied from 1 to 8 hours, dependent on the data subset and architecture used. Models trained for between 10 and 30 epochs. Fine-tuning on the clean deathmatch dataset took 15 minutes per epoch, typically requiring 12 to 32 epochs.

VI. EVALUATION

This section evaluates the agent in two ways: 1) Measuring the score it achieves relative to both human players and the built-in rules-based bot. 2) Assessing the ‘humanlike-ness’ of the agent’s play style, done both qualitatively and also by quantitatively analysing its map coverage.

Gameplay examples are shared at: <https://youtu.be/KTY7UhjIMm4>. Clips were selected by running the strongest agent on each game mode and setting for five minutes, and

selecting a one minute segment from each which showcases the agent’s skill. The video further includes: illustrations of the the agent’s common failures, a longer unedited clip of the agent on the medium setting, a longer unedited clip of the agent navigating in an empty map. Code to run the agent is provided in the [repo](#).

A. Hyperparameter Search

We trained multiple versions of the agent on the online dataset varying several hyperparameters: 1) Adding an extra LSTM layer (256 units) after the convolutional LSTM. 2) Applying dropout to recurrent connections. 3) Training on different subsets of the data; as well as training over the full dataset, we also considered only sequences where the AK47 was equipped, and only sequences where the AK47 or M4A1 was equipped. 4) We optionally undersampled sequences where a player did not score a kill (‘non-scoring’).

B. Agents & Baselines

The best performing agent trained only on AK47 data, undersampled non-scoring sequences with a probability of 0.4, used recurrent dropout, and had no extra LSTM layer. This is termed ‘**online agent**’. This online agent was then fine-tuned on the clean datasets, producing ‘**fine-tuned dm agent**’ and ‘**fine-tuned aim agent**’.

We include several baselines. 1) **Built-in Bot (easy)** – the bots played against in the deathmatch easy setting. 2) **Built-in Bot (medium)** – the bots played against in the deathmatch medium setting. 3) **Human (Non-gamer)** – someone with little experience playing games. 4) **Human (Casual gamer)** – a regular player of video games, with a small amount (<100 hours) of CSGO experience. 5) **Human (Strong)** – a player ranked in the top 10% of regular CSGO players (‘DMG’ rank). All humans play at full 1920×1080 resolution, and are assessed over 5 minutes (aim train mode) and 10 minutes (deathmatch modes) of play. Longer periods resulted in fatigue and decreased performance.

C. Main Results

Table III displays results of the best performing agents. We report two metrics; kills-per-minute (KPM) and kill/death ratio (K/D) – higher is better for both. For each game mode and setting we report the mean and one standard error over eight episodes of 10 minutes, restarting the game at least three times within these eight episodes to randomise the opponents.

Aim train mode: The fine-tuned aim agent’s performance is in line with the casual gamer’s (note K/D is not applicable to this mode since the agent cannot take damage). The agent demonstrated good accuracy and recoil control, prioritising enemy targets sensibly, and anticipating their motion. Aim train mode required less clean training data than deathmatch for good performance (45 minutes vs 190 minutes) – this is because it is a visually simpler environment and requires behaviour over short time horizons only (table II). The online agent was able to somewhat generalise to the new environment, although it was unfamiliar with the specific movement patterns of enemies.

TABLE III
 MAIN RESULTS. METRICS ARE KILLS-PER-MINUTE (KPM) AND KILLS/DEATH RATIO (K/D). HIGHER IS BETTER. MEAN \pm ONE STD. ERROR.

	Aim Train KPM	Deathmatch					
		Easy		Medium		Human	
	KPM	K/D	KPM	K/D	KPM	K/D	
Dataset used							
Online dm	4.31 \pm 0.20	3.47 \pm 0.12	2.70 \pm 0.26	2.23 \pm 0.26	1.04 \pm 0.06	0.68 \pm 0.13	0.22 \pm 0.03
Online dm + Clean aim/dm	26.86 \pm 0.39	5.06 \pm 0.31	3.87 \pm 0.24	3.72 \pm 0.25	2.09 \pm 0.19	1.43 \pm 0.17	0.59 \pm 0.09
Baselines							
Built-in Bot (easy)	–	2.11	1.00	–	–	–	–
Built-in Bot (medium)	–	–	–	2.41	1.00	–	–
Human (Non-gamer)	14.32	4.25	1.80	2.38	0.90	0.75	0.27
Human (Casual gamer)	26.21	4.20	4.20	3.51	2.48	1.64	0.64
Human (Strong CSGO player)	33.21	14.00	11.67	7.80	4.33	4.27	2.34

Deathmatch mode: The fine-tuned dm agent outperforms the built-in bot, both on easy and medium settings, roughly matching the performance of the casual gamer. The agent navigates around the majority of the map well, identifying and reacting to enemies reliably, and distinguishing them from teammates. It also chooses sensible moments to reload. Moving from the online agent to the fine-tuned agent improves KPM by around 45%, showing the importance of converging on a single high-skill policy. However, a performance gap remains between the agent and strong human.

D. Humanlike Assessment

This paper is not exclusively interested in the score-based performance of the agent. Another key goal of agent design in video games is to build humanlike agents. One of the advantages of using behavioural cloning is that the agent naturally adopts humanlike traits. Measurement of this humanlike quality is less straightforward, and we qualitatively discuss traits observed during testing – behaviours that are common to human players, but not seen in the built-in bots. The agent’s map coverage is then quantitatively analysed. Future work might provide further analysis through Turing-style tests with human observers and players.

Humanlike traits: The agent’s mouse movement mimics that of a human, pausing mid-turn as if the mouse had reached the edge of the mouse pad. The agent’s navigation is quite humanlike, often running along ledges or jumping over obstacles. In certain areas it will jump to spot an occluded area. The online agent sometimes exhibits playful quirks, such as firing at chickens or ‘bunny-hopping’. The fine-tuned dm agent occasionally employs higher-skill behaviours, such as moving behind cover when reloading, or strafing during fire-fights.

Non-humanlike traits: The agent makes several mistakes humans do not. It’s memory is poor – if an enemy disappears behind cover it quickly forgets about it. (This still occurred after adding an extra LSTM layer.) It also does not pick up on ‘second-order clues’ about where enemies may be (e.g. teammates firing in some direction). It is poor at aiming vertically. In one region of the map that is rarely visited by human players (bottom left in figure 3) its navigation is poor. Occasionally (once in 10 minutes) the agent gets into a position it can’t recover from.

There are several more understandable limitations. The agent only receives the image as input, so has no audio clues that humans react to (shots being fired, or enemy footsteps). It also rarely reacts to red damage bar indicators.

Quantitative Map Coverage Analysis. To quantitatively assess the similarity of the agent to human play, we track the x & y coordinates of the agent playing on the medium deathmatch mode for 100 minutes. We discretise the map on a 60×60 grid, and count the amount of time spent in each box. This distribution is compared to human play in the online and clean datasets, as well as the built-in bot.

We consider two versions of the agent; one trained over the *full* dataset and one subsequently fine-tuned on the clean dm dataset. Figure 3 shows map coverage heatmaps for each policy. One first observes that the agent’s histograms mimic the routes taken by human players more closely than the built-in bots. Secondly, the online dm agent’s coverage is most similar to that of the online dataset, while the coverage of the agent fine-tuned on the clean dataset is most similar to the clean dm dataset. These observations are quantified in table IV, where similarity between pairs of distributions is computed via the Earth mover’s distance.

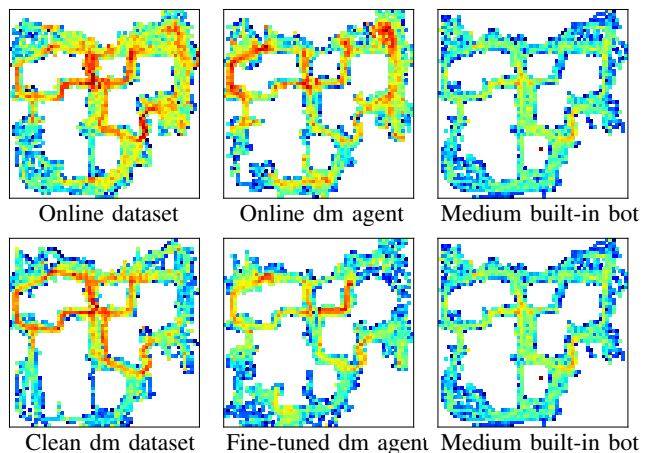


Fig. 3. Map coverage heatmap for agent, built-in bot, and human datasets. Note the agent’s coverage mimics the data it’s trained on. Quantitatively this is captured by the ‘Earth mover’s distance’ between each distribution as in table IV.

TABLE IV
DIFFERENCE BETWEEN DISTRIBUTIONS SHOWN IN FIGURE 3. EARTH
MOVER’S DISTANCE, LOWER IS MORE SIMILAR.

	Online dm	Clean dm	Online agent (full)	Fine-tuned agent	Built-in Bot
Vs. Online dm	0.000	0.977	0.447	0.941	0.624
Vs. Clean dm	0.970	0.000	0.533	0.349	0.643

E. Ablations on Data Size and Pre-training

Ablations were run to study the benefit of pre-training on the online dataset, compared with training models from scratch on the clean deathmatch dataset. These experiments were applied to the agent trained over the *full* online dataset. Using three choices of weight initialisation, {random, ImageNet, online dataset pre-training}, we fine-tuned on varying amounts of clean demonstration data; {1, 2, 3} hours. Results are shown in figure 4.

Whilst using the weights from ImageNet improved over random initialisations, there is a large benefit to using weights pre-trained on the online dataset – extrapolating the curves in the figure suggest around 20 hours of clean expert data would be needed to match the performance of the pre-trained model fine-tuned on just 2 hours of clean expert data.

VII. DISCUSSION & CONCLUSION

This paper presented an AI agent that plays the video game CSGO from pixels, matching the skill-level of a casual human gamer. It is the first effort to tackle a modern FPS game, and is amongst the largest-scale works in behavioural cloning in any genre to date.

Whilst the AI community has historically focused on real-time-strategy games such as Starcraft, we see CSGO as an equally worthy test-bed, providing its own unique mix of control, navigation, teamwork, and strategy. Its large and long-lasting player base, as well as similarity to other FPS titles, means AI progress in CSGO is likely to attract broad interest, and also suggests tangible value in developing strong, humanlike agents.

Although an inconvenience to researchers, CSGO’s inability to be simulated at scale arguably creates a challenge more representative of those in the real-world, where RL algorithms can’t always be run from a blank state, and even evaluating agents can be a costly process. This paper has defined several game modes of varying difficulty, and had a first attempt at solving them with behavioural cloning. We share our code and datasets to encourage other researchers to partake in this environment’s challenges.

There are many directions in which our research might be extended, such as applying more advanced methods from imitation learning or offline RL, or integrating with reward-based learning. More ambitiously, there’s the challenge of taking on CSGO’s full competitive mode – we see our paper as a step toward that AI milestone.

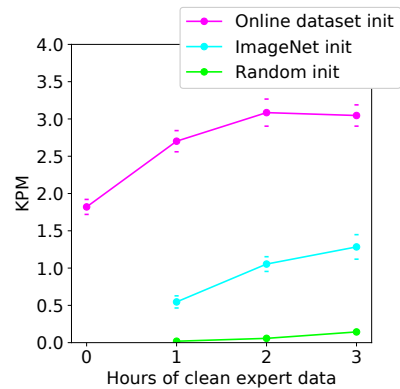


Fig. 4. Ablation of initialisation point and amount of clean expert data. Medium difficulty deathmatch over 10 episodes of 10 minutes. Mean \pm one standard error.

ACKNOWLEDGMENTS

Thanks to Bang Xiang Yong, Grace E. Lee and Charlotte Morrill for providing the baselines. Also to Sam Devlin for discussion, and anonymous reviewers for useful feedback that has improved the paper.

REFERENCES

- [1] R. Agarwal, D. Schuurmans, and M. Norouzi. An Optimistic Perspective on Offline Reinforcement Learning. *ICML*, 2020.
- [2] P. Bakker and Y. Kuniyoshi. Robot See, Robot Do : An Overview of Robot Imitation. *AISB workshop on learning in robots and animals*, (May 1996), 1993.
- [3] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, A. Lefrancq, S. Green, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen. DeepMind Lab. pages 1–11, 2016.
- [4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. De Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv*, 2019. ISSN 23318422.
- [5] B. Bukaty and D. Kanne. Using Human Gameplay to Augment Reinforcement Learning Models for Crypt of the NecroDancer. *ArXiv*, 2020.
- [6] Z. Chen and D. Yi. The game imitation: Deep supervised convolutional networks for quick video game AI. *arXiv*, 2017. ISSN 23318422.
- [7] N. Cole, S. J. Louis, and C. Miles. Using a genetic algorithm to tune first-person shooter bots. *Proceedings of the 2004 Congress on Evolutionary Computation, CEC2004*, 1:139–145, 2004. doi: 10.1109/cec.2004.1330849.
- [8] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. Datasets for Data-Driven Reinforcement Learning. pages 1–13, 2020. URL <http://arxiv.org/abs/2004.07219>.
- [9] B. Gorman and M. Humphrys. Imitative learning of combat behaviours in first-person computer games. *Proceedings of CGAMES 2007 - 10th International Conference on Computer Games: AI, Animation, Mobile, Educational and Serious Games*, pages 85–90, 2007.
- [10] B. Gorman, C. Thureau, C. Bauckhage, and M. Humphrys. Believability testing and Bayesian imitation in interactive computer games. *Lecture Notes in Computer Science (including subseries*

- Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*), 4095 LNAI(May 2014):655–666, 2006. ISSN 16113349. doi: 10.1007/11840541_54.
- [11] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. MineRL: A large-scale dataset of minecraft demonstrations. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2442–2448, 2019. ISSN 10450823. doi: 10.24963/ijcai.2019/339.
- [12] T. S. Hardenstein. “Skins” in the Game: Counter-Strike, Esports, and the Shady World of Online Gambling. *World*, 419(2015): 117–137, 2006.
- [13] J. Harmer, L. Gisslén, J. del Val, H. Holst, J. Bergdahl, T. Olsson, K. Sjö, and M. Nordin. Imitation learning with concurrent actions in 3d games. *arXiv*, pages 1–8, 2018. ISSN 23318422.
- [14] T. Hester, T. Schaul, A. Sendonaris, M. Vecerik, B. Piot, I. Osband, O. Pietquin, D. Horgan, G. Dulac-Arnold, M. Lanctot, J. Quan, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep q-learning from demonstrations. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3223–3230, 2018.
- [15] P. Hingston. The 2K BotPrize. *IEEE Symposium on Computational Intelligence & Games*, 2009. doi: 10.1109/cig.2009.5286502.
- [16] S. Hladky and V. Bulitko. An evaluation of models for predicting opponent positions in first-person shooter video games. *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, pages 39–46, 2008. doi: 10.1109/CIG.2008.5035619.
- [17] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, and J. Z. Leibo. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 2019.
- [18] A. Kanervisto, J. Pussinen, and V. Hautamaki. Benchmarking End-to-End Behavioural Cloning on Video Games. *IEEE Conference on Computational Intelligence and Games, CIG, 2020-Augus:558–565*, 2020. ISSN 23254289. doi: 10.1109/CoG47356.2020.9231600.
- [19] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. *IEEE Conference on Computational Intelligence and Games, CIG, 2016*. ISSN 23254289. doi: 10.1109/CIG.2016.7860433.
- [20] G. Lample and D. S. Chaplot. Playing FPS Games with Deep Reinforcement Learning. *AAAI*, 2017. doi: arXiv:1609.05521v2. URL <http://arxiv.org/abs/1609.05521>.
- [21] M. Laskey, J. Lee, W. Hsieh, R. Liaw, J. Mahler, R. Fox, and K. Goldberg. Iterative Noise Injection for Scalable Imitation Learning. *ArXiv preprint*, (CoRL):1–14, 2017. URL <http://arxiv.org/abs/1703.09327>.
- [22] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv*, 2020. ISSN 23318422.
- [23] I. Makarov, D. Savostyanov, B. Litvyakov, and D. I. Ignatov. Predicting winning team and probabilistic ratings in “Dota 2” and “Counter-strike: Global offensive” video games. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10716 LNCS(January):183–196, 2018. ISSN 16113349. doi: 10.1007/978-3-319-73013-4_17.
- [24] V. Mnih, I. Antonoglou, A. K. Fidjeland, D. Wierstra, H. King, M. G. Bellemare, S. Legg, S. Petersen, M. Riedmiller, C. Beattie, A. Graves, A. Sadik, K. Kavukcuoglu, G. Ostrovski, J. Veness, A. A. Rusu, D. Silver, D. Hassabis, and D. Kumaran. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN 0028-0836. doi: 10.1038/nature14236. URL <http://dx.doi.org/10.1038/nature14236>.
- [25] S. Ontañón, J. L. Montaña, and A. J. Gonzalez. Expert Systems with Applications A Dynamic-Bayesian Network framework for modeling and evaluating learning from observation. *Expert Systems with Applications*, 2014.
- [26] V. M. Petrovic. Artificial Intelligence and Virtual Worlds-Toward Human-Level AI Agents. *IEEE Access*, 6:39976–39988, 2018. ISSN 21693536. doi: 10.1109/ACCESS.2018.2855970.
- [27] F. Reer and N. C. Krämer. Underlying factors of social capital acquisition in the context of online-gaming: Comparing World of Warcraft and Counter-Strike. *Computers in Human Behavior*, 36:179–189, 2014. ISSN 07475632. doi: 10.1016/j.chb.2014.03.057. URL <http://dx.doi.org/10.1016/j.chb.2014.03.057>.
- [28] S. Ross, G. J. Gordon, and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *AISTATS*, 2011. URL http://www.ri.cmu.edu/pub/_files/2011/4/Ross-AISTATS11-NoRegret.pdf.
- [29] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems 28*, pages 802–810, 2015. ISSN 10495258.
- [30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN 14764687. doi: 10.1038/nature16961. URL <http://dx.doi.org/10.1038/nature16961>.
- [31] M. Tan and Q. V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. *36th International Conference on Machine Learning, ICML 2019*, 2019.
- [32] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019. ISSN 14764687. doi: 10.1038/s41586-019-1724-z. URL <http://dx.doi.org/10.1038/s41586-019-1724-z>.
- [33] M. Wydmuch, M. Kempka, and W. Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 11(3):248–259, 2019. ISSN 24751510. doi: 10.1109/TG.2018.2877047.