A Comparison of Behavior Cloning Methods in Developing Interactive Opposing-Force Agents

Logan Lebanoff, Nicholas Paul, Christopher Ballinger, Patrick Sherry,
Gavin Carpenter, Charles Newton
Soar Technology, Inc.
Orlando, FL
{logan.lebanoff, nicholas.paul, cballinger, patrick.sherry, gavin.carpenter, charles.newton}@soartech.com

Abstract

Modern modeling and simulation environments, such as commercial games or military training systems, frequently demand interactive agents that exhibit realistic and responsive behavior in accordance with a predetermined specification, such as a storyboard or military tactics document. Traditional methods for creating agents, such as state machines or behavior trees, necessitate a significant amount of effort for developing state representations and transition processes through manual knowledge engineering. On the other hand, newer techniques for behavior generation, such as deep reinforcement learning, require a vast amount of training data (centuries in many cases), and there is no guarantee that the generated behavior will align with intended objectives and courses of action. This paper examines the application of behavior cloning approaches in designing interactive agents. In our approach, users start by defining desired behavior through straightforward methods such as state machine models or behavior trees. Behavior cloning methods are then used to transform ground-truth trajectory data sampled from these models into differentiable policies that are further refined through engagement with interactive game environments. This method results in improvements in training results when compared on dimensions of task performance and stability of training.

Introduction

High-fidelity training, gaming, and simulation software products often include AI-enabled opponents that provide entertaining, competitive, and realistic interaction opportunities. However, developing these opponent forces (OPFOR) is a resource-intensive process. Common techniques used in industry to implement AI-driven behaviors include the use of behavior trees and reinforcement learning algorithms. Each technique involves trade-offs: behavior trees are relatively straightforward to implement and visualize but involve a large amount of manual knowledge encoding and do not easily generalize to unanticipated situations. Reinforcement learning approaches exhibit stronger generalization capabilities but are difficult for behavior designers to control, visualize, or explain and most use cases require enormous amounts of training data. In this paper, we propose behavior cloning as a best-of-both-worlds approach where behavior designers first develop simple and authorable state machines that provide a rough approximation of the desired agent behavior. Behavior trajectories generated by these state machines executing in an interactive game environment are used as training data for deep reinforcement offline policy learning approaches. Finally, pretrained policies are transitioned to an online learning approach where they are presented with the same environment but under a deep reinforcement learning regime with an open-ended reward function based on objective task performance.

Behavior cloning approaches are evaluated based on performance in terms of learning stability (standard deviation and interquartile range) and task performance in a military-oriented kinetic engagement environment based on USC Institute for Creative Technologies' Rapid Integration & Development Environment (RIDE; Hartholt et al. 2021) platform. We test several network architectures, including multilayer perceptrons (MLP) with and without frame-stacking and Transformers (Vaswani et al. 2017). Our contributions are as follows.

- We present an investigation into behavior cloning approaches in a military-oriented simulation environment. In these types of environments, it is often crucial for agents to demonstrate high levels of performance while also adhering to realistic, predefined behaviors.
- We find that behavior cloning with Transformer models leads to agents that often outperform human-authored state machines. Further, the agents retain many desired behaviors that were defined in the state machines.

Related Work

Deep Reinforcement Learning

The concept of machines automatically learning desired sequential behavior in an environment based on rewarding desired behavior and optimizing for maximizing the expected total return of reward was first explored by Bellman (1957); Howard (1960); Andreae (1963); Michie (1963); and Widrow, Gupta, and Maitra (1973). Deep reinforcement learning (DRL) (Li 2017; François-Lavet et al. 2018; Li 2018) is a highly general learning method that has shown successful performance in diverse environments such as self-driving vehicles (Goswami, Howley, and Mannion

2019), stock trading (Xiong et al. 2018), and text summarization (Paulus, Xiong, and Socher 2018; Stiennon et al. 2020).

Imitation Learning

An open question is the best approach for tailoring these general learning methods for improved performance within a specific environment or domain (Xing et al. 2021; Wang et al. 2022; Singireddy, Jha, and Velasquez 2023). One tradeoff that exists is the application of learning from previous demonstrations in the environment (Pomerleau 1991; Ho and Ermon 2016) versus learning through real-time interactions in the environment (Saad 1999). Offline learning can learn more effectively using previously collected datasets to more rapidly learn behavior (Abbeel and Ng 2004). However, online methods excel in situations where the distribution of states is dynamic (Ross and Bagnell 2010). Approaches such as offline-to-online deep reinforcement learning (Uludağ et al. 2013) have been explored that can take advantage of the ability of offline learning methods to rapidly learn a problem while still relying on online learning to handle environments exhibiting non-stationarity. A simple form of learning from expert demonstrations is *behavior cloning*, first applied by Pomerleau (1991) to develop control algorithms that artificially mimic expert demonstrations of land vehicle drivers as a part of DARPA's Strategic Computing initiative program (Stefik 1985; Roland, Shiman, and others 2002). In behavior cloning, a model directly learns to map environment states to agent actions based on previously performed executions within the environment.

Reinforcement Learning with Transformers

In DRL, decision policies and other auxiliary functions (e.g., value functions) that support decision-making are represented as neural networks. Recently, Chen et al. (2021) proposed the utilization of the Transformer architecture (Vaswani et al. 2017) to model decision policies that are conditioned on past agent states / actions and the desired return from a reward function. These decision Transformers (DTs) outperform competing approaches on standard Atari (Bellemare et al. 2013) and D4RL (Fu et al. 2020) benchmarks in an offline learning setting. Other recent extensions to the DT architecture in offline settings include the Trajectory Transformer (Janner, Li, and Levine 2021), which uses Transformers to produce learned distributions over potential agent trajectories, leverages a beam-search decoder as a planning algorithm, and shows advantages in sparse-reward and long-horizon tasks.

Transformers can also be applied in multi-modal decision domains, as demonstrated by the generalist agent proposed by Reed et al. (2022) which uses the same architecture and weights to perform functionalities such as playing Atari games, acting as a chatbot, serving as a robotic arm control system, and performing image captioning. Similarly, (Baker et al. 2022) applies Transformers to learn with large-scale observational data in the form of unlabeled demonstration videos to improve sample efficiency, particularly with long and complex tasks that would be difficult to discover without human demonstration. Micheli, Alonso, and Fleuret (2022) propose a Transformer-based world model that can be trained to generate simulations of the environment, which makes model-based Reinforcement Learning (RL) algorithms more sample-efficient.

Games and Simulations

Research in reinforcement learning has also been applied towards training agents for military simulation environments (Stein and Kobrick 1984), with an emphasis on opposing-force agents (Boron and Darken 2020). Opposing-force agents are agents trained to specifically act as AI opponents in simulation environments that replicate the actions and behaviors of real-world adversaries (Army 2017; Bonasso 1988; White 1986). The development of opposing-force agents can provide important insights into the behavior and strategies of real-world adversaries in military scenarios. For example, the Joint Semi-Automated Forces (JSAF) system is a widely adopted military simulation environment that has been used in many multi-national training exercises (Hassaine et al. 2006). In these systems, opposing-force agents are trained to simulate the behavior and tactics of enemy forces, allowing trainees to improve their decision-making processes in simulated military scenarios (Turnitsa, Blais, and Tolk 2022). It is vital to improve the realism of these opposing-force agents, as higher fidelity training simulations have been shown to lead to better real-world performance (Seymour et al. 2002; Ragan et al. 2015; Whitmer, Ullman, and Johnson 2019).

Behavior Cloning Methodology

Behavior cloning (Pomerleau 1991) is a supervised learning method for creating policies by using expert demonstrations. Our evaluation focuses on the effectiveness of behavior cloning in environments where only a simple, low-quality model is available (e.g., a rough computational model produced by an expert) but an online environment exists in which these models can be automatically evaluated and improved through deeper refinement via reinforcement learning. This paper evaluates two different classes of behavior cloning approaches:

1. A cloning approach that first trains a multi-layer perceptron (MLP; see Hastie et al. 2009) to represent the actions of a finite state machine and then further refines this MLP policy via proximal-policy optimization (PPO; Schulman et al. 2017).

2. A cloning approach that trains a Generative Pretrained Transformer (GPT; see Radford et al. 2019) to reproduce the actions of a finite state machine and also refines this GPT policy via PPO.

These approaches are evaluated in a "kill-box" domain (Broyles et al. 2022), a common military fire control measure where threats are contained in terms of geography, tactics, rules of engagement, or other constraints and forces can freely engage. Figure 1 presents a high-level overview of our architecture.

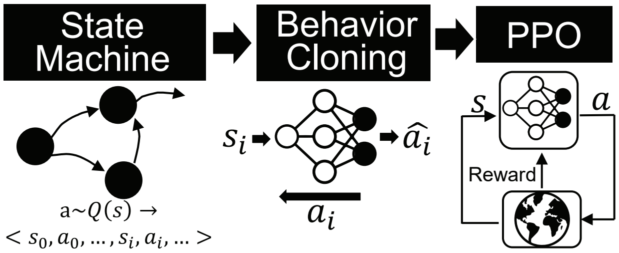

Figure 1: Behavior cloning pipeline.

Terminology

We consider a set of states S where each $s \in S$ represents an agent state within an environment. Agents can take actions $a \in A$ and actions are assumed to take place in a discrete fashion indexed by timestamp t. Agent actions are determined by either a finite state machine Q or a neural-policy π, both of which map an agent state s_t to action a_t. Environments emit a reward for the agent $r(s_i, a_i)$, and the expected return associated with a state is denoted $V(s_t)$.

State Machine

Our architecture utilizes finite state machines (FSM) as an initial source of supervised demonstration data. State machines are assumed to not necessarily be expert-level, but simply execute relevant operations that sometimes achieve reward in the environment. Within the kill-box domain, the state machine fired its weapon if a threat was in front of the controlled simulation entity, maneuvered towards a nearby threat if one was to the left or right, and moved forward until the forward direction was blocked, in which case the entity moved backward.

MLP

We use a multi-layer perceptron (MLP) as the agent's base network architecture. The MLP receives the agent state $\mathbf{s_t}$ as input and produces an intermediate state representation $\mathbf{h}$:

$$\mathbf{h}_0^b = \mathbf{s}_t \tag{1}$$

$$\mathbf{h}_i^b = \text{ReLU}(\mathbf{W}_i^b \mathbf{h}_{i-1}^b + \mathbf{b}_i^b) \quad \text{for } i = 1, \ldots, N \tag{2}$$

where $\mathbf{h}_i^b$ is the output of the ith layer in the base network, $\mathbf{W}^b$ and $\mathbf{b}^b$ are the weights and biases, ReLU is the rectified linear unit activation function, and N is the number of layers.

This intermediate representation is then fed through additional layers to compute the agent's policy π and the expected return V (see *PPO* section for the usage of V):

$$\mathbf{h}_0^a = \mathbf{h}_N^b \tag{3}$$

$$\mathbf{h}_0^v = \mathbf{h}_N^b \tag{4}$$

$$\mathbf{h}_i^a = \text{ReLU}(\mathbf{W}_i^a \mathbf{h}_{i-1}^a + \mathbf{b}_i^a) \quad \text{for } i = 1, \ldots, N \tag{5}$$

$$\mathbf{h}_i^v = \text{ReLU}(\mathbf{W}_i^v \mathbf{h}_{i-1}^v + \mathbf{b}_i^v) \quad \text{for } i = 1, \ldots, N \tag{6}$$

$$\pi(\mathbf{a}|\mathbf{s}) = \text{softmax}(\mathbf{W}_{N+1}^a \mathbf{h}_N^a + \mathbf{b}_{N+1}^a) \tag{7}$$

$$V(s) = \mathbf{W}_{N+1}^v \mathbf{h}_N^v + \mathbf{b}_{N+1}^v \tag{8}$$

where $\mathbf{h}_i^a$ and $\mathbf{h}_i^v$ are the outputs of the ith layer in the policy and value networks respectively, $\mathbf{W}^a$, $\mathbf{W}^v$, $\mathbf{b}^a$, and $\mathbf{b}^v$ are the weights and biases for the policy and value networks.

Frame Stacking

We also experiment with frame-stacking to determine whether past states can help the agent learn more effectively. Frame stacking keeps track of the most recent states $\langle s_{t-n}..., s_{t-1}, s_t \rangle$, where n is the number of frames to stack, and concatenates them together to be used as input to the agent. This replaces Eq (1) with the following:

$$\mathbf{h}_0^b = \text{Concat}(\mathbf{s}_{t-n}..., \mathbf{s}_{t-1}, \mathbf{s}_t) \tag{9}$$

Mnih et al. achieved state-of-the-art results on Atari games by using deep Q-learning and stacking the last 4 frames of history together to be used as input to their agents (Mnih et al. 2013; 2015).

Transformer

Recently, Transformers (Vaswani et al. 2017) have been used in domains such as text generation (Radford et al. 2019) and computer vision (He et al. 2022) to effectively model semantic knowledge in high-dimensional space. Transformers have also been used successfully in offline RL-based sequence modeling problems by conditioning on previous states, actions, and returns (Chen et al. 2021; Janner, Li, and Levine 2021).

Similarly, in this work, a history of the agent's observations is used to generate a sequence of past states $\langle s_{t-n}..., s_{t-1}, s_t \rangle$. This sequence is encoded using a decoder-only Transformer (specifically GPT-2 Small; Radford et al. 2019), with randomly-initialized weights. A Transformer block uses a multi-head attention mechanism followed by a feed-forward network to process a series of states.

$$\text{SelfAttn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{10}$$

$$\text{MultiHead}(Q, K, V) = \odot(head_1, \ldots, head_h)W^O \tag{11}$$

$$head_i = \text{SelfAttn}(QW_i^Q, KW_i^K, VW_i^V) \tag{12}$$

$$\text{TransformerBlock}(h) = \text{LayerNorm}(u + \text{FFN}(u)) \tag{13}$$

$$u = \text{LayerNorm}(h + \text{MultiHead}(h, h, h)) \tag{14}$$

where d_k is the dimensionality of the keys K and values V in the self-attention mechanism, $\odot$ is the concatenation operator, and h is the input to the Transformer block. The full Transformer is a series of Transformer blocks.

Finally, Eqs. (1 - 2) are replaced with the following.

$$\mathbf{x} = \mathbf{s}_{t-n}..., \mathbf{s}_{t-1}, \mathbf{s}_t \tag{15}$$

$$\mathbf{h}_N^b = \text{Transformer}(\mathbf{x}) \tag{16}$$

which replaces the base feed-forward network layers with a Transformer.

Behavior Cloning

After choosing a base network architecture – either MLP, MLP with frame stacking, or Transformer – the network is trained using behavior cloning. First, a large number of agent trajectories are collected by running the FSM in the kill-box environment. Each trajectory contains the state and action taken for every step. In our experiments, we collect data over 1000 episodes with 1000 steps per episode, totaling 1M steps. Next, these collected trajectories are used as training data for the network. The network is trained to predict the next action taken at each step given the state (or set of recent states), in a supervised manner. It is trained to minimize the cross-entropy loss between the predicted action and the FSM's action.

$$\mathcal{L}_{BC} = -\frac{1}{C}\sum_{i=1}^{C} a_i \log \pi_i \tag{17}$$

where C is the number of possible actions and a_i is the FSM's label given to the action.

Effective training using this process results in a policy π that models or "clones" the behavior of the FSM. PPO is used to further refine this policy using online learning.

PPO

Proximal Policy Optimization (PPO; Schulman et al. 2017) is used to further refine the policy π via online learning by exposing it to the kill-box environment and updating the policy via policy gradient by maximizing the expected reward. The policy's loss function seeks to maximize the advantage A, which is the incremental reward of taking the next action from the current state.

$$\mathcal{L}_\pi(\theta) = -E_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \cdot A(s_t, a_t)\right] \tag{18}$$

$$\text{where } \|\theta - \theta_{\text{old}}\| \leq \epsilon \tag{19}$$

$$A(s,a) = r(s,a) + \gamma V(s') - V(s) \tag{20}$$

where π_θ is the current policy and $\pi_{\theta_{\text{old}}}$ is a previous version of the policy, $r(s,a)$ is the reward for taking action a, γ is the discount factor, and s' is the next state after taking action a.

The value function's loss is computed as the Mean Squared Error.

$$\mathcal{L}_V(\theta) = \frac{1}{N}\sum_{t=1}^{N}(V_\theta(s_t) - y_t)^2 \tag{21}$$

where y_t is the target expected return for state s_t, and N is the total number of training samples.

The final loss function is the sum of the policy and value function losses.

$$\mathcal{L}_{PPO}(\theta) = \mathcal{L}_\pi(\theta) + \mathcal{L}_V(\theta) \tag{22}$$

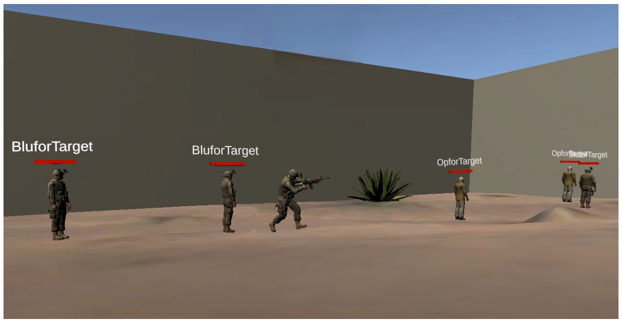

Figure 2: Kill-box environment. The agent (center) must fire at enemy targets while avoiding friendly targets.

Event	Reward
Shooting enemy	+1
Shooting ally	-4
Firing gun and not hitting anything	-2
Reloading with shots fired	+1
Reloading with no shots fired	-2
Walking into boundary walls	-10

Table 1: Rewards used in kill-box experiments

Evaluation

The kill-box experimental setup consists of 100 stationary soldiers – half are enemy targets and half are friendly targets – in a space enclosed by 4 walls (see Figure 2). The agent is rewarded for firing at enemy targets and penalized for firing at anything else (see Table 1 for full reward function). We test the following methods. *FSM* is a manually-created state machine. *MLP* is a baseline using a feed-forward network trained only with PPO (no behavior cloning). *MLP+BC* is our method of using a feed-forward network that is trained to clone *FSM's* behavior and then refined using PPO. *MLP+BC+FS* uses frame stacking in addition. *Transformer+BC* uses a decoder-only Transformer in place of a feed-forward network.

We evaluate the methods from two perspectives: 1) agent task performance and 2) stability of training. However, different training approaches tend to have different hyperparameters, which greatly influence a model's learning performance. In order to more effectively compare methodologies, we perform a sweep over the hyperparameter values for each methodology. Current reinforcement learning algorithms are notoriously unstable, difficult to train, and sensitive to hyperparameter values (Sprague 2015; Henderson et al. 2018; Zhang et al. 2021), which necessitates searching over the possible hyperparameter values to obtain positive results (Paine et al. 2020; Hauser 2021). There are a combinatorially explosive number of hyperparameter combinations for each training approach, that makes exhaustively searching for the best set of hyperparameters intractable. Instead, we use the *Weights & Biases* (Biewald 2020) library to explore the values of hyperparameters that have the greatest influence over training performance and outcomes us-

Method	Mean	Med.	Max	σ
MLP	-57.26	-17.01	78.60	129.08
MLP + BC	-154.84	18.18	144.38	1029.28
MLP + BC + FS	8.10	14.33	391.76	**97.18**
Transformer + BC	**59.55**	**26.69**	**697.18**	110.73
FSM	36.52	28.55	255.51	39.11

Table 2: Agent performance comparison (measured by reward score over 50 runs) in kill-box environment.

ing Bayesian optimization (Močkus 1975). This enables us to explore a wider set of training parameters in a shorter amount of time. We perform hyperparameter sweeps for each of methodology for 50 runs each, and computed the mean, median, and maximum performance over the 50 runs for each methodology (Table 3).

Results

Table 2 presents the performance of each method, measured as the reward score. The *MLP* baseline using only PPO (no behavior cloning) gives median and max scores that are significantly lower than the methods using behavior cloning. This shows that online reinforcement learning approaches alone are insufficient for modeling interactive agents in our domain, whose expected behaviors may be too difficult to learn from scratch. Behavior cloning allows the agent to learn from the state machine's demonstrations and helps to improve the overall performance of the agent. The methods that use behavior cloning in combination with PPO outperform the *MLP* baseline by a large margin, demonstrating the importance of incorporating prior knowledge and experience into the learning process.

Next, we assess the value of frame-stacking by comparing *MLP+BC* to *MLP+BC+FS*. *MLP+BC+FS* has significantly higher mean and max scores (albeit slightly lower median score). In addition, *MLP+BC+FS* has a much lower standard deviation. The large difference in mean and standard deviation are mostly driven by a few outlier runs from *MLP+BC*, which were as low as -7000. Prior work has shown that it can be challenging to transition from an offline to an online RL setting (Rajeswaran et al. 2017; Fujimoto, Meger, and Precup 2018), which we hypothesize is the case with *MLP+BC*. Our results indicate that including a history of past observations, using frame stacking, can lead to better performance and higher stability of training. We hypothesize that the contextual information from previous frames prevents the agent from going too off-course since the agent can learn to use the past frames to better understand its current situation.

The best-performing method is *Transformer+BC*, with the largest mean, median, and max scores. The Transformer has shown to be a powerful method for sequence modeling, outperforming methods using long short-term memory (LSTM) or feed-forward networks (i.e. MLP) in many domains (Vaswani et al. 2017; Dosovitskiy et al. 2021; Arnab et al. 2021). Transformers can more effectively model sequences of data – in our case, agent observations and actions – using an attention mechanism. Additionally, prior work has shown that Transformers with a longer context length outperform simple frame-stacking on Atari games (Chen et al. 2021). Our results in a military-oriented kinetic environment seem to follow this trend. The mean and max scores of *Transformer+BC* are higher than those of the original *FSM* from which it was copied, indicating our method of refining a behavior-cloned model through PPO can lead to agents that surpass the original state machine.

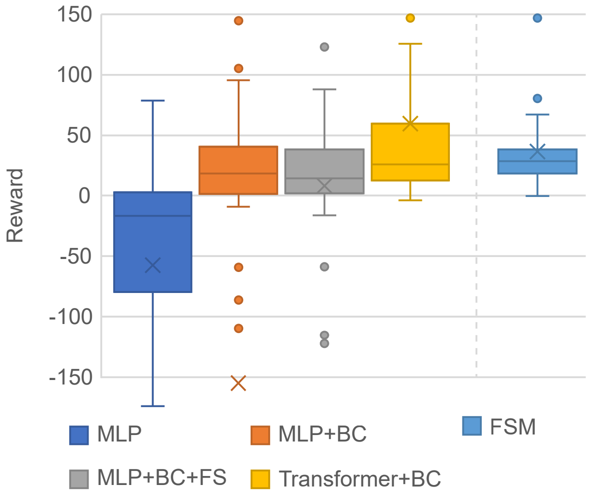

Figure 3: Box plot comparing the reward scores of each method. Outliers exist that are greater than 150 and less than -150, but the plot has been truncated for readability.

Figure 3 presents our results with interquartile ranges. Again, we see the Transformer outperforms other methods, including the FSM. The Transformer does however have slightly lower median score, along with a higher standard deviation and a larger interquartile range, indicating there is a greater amount of variability between runs. Nevertheless, the Transformer presents a greater likelihood of producing exceptional agents with rewards exceeding 50, given multiple runs are conducted. Interestingly, *MLP+BC* has similar interquartile ranges to *MLP+BC+FS*, whereas the standard deviations in Table 2 between them are very different. This shows that most runs without frame stacking will be in the typical range, but it will be more likely to produce an unstable, degenerate agent.

Another benefit from behavior cloning is that the resulting agent is of higher quality while still retaining much of the desired behavior defined in the state machine. For example, in the FSM used for these experiments, a "reload" state is activated whenever the agent transitions to the moving forward state from any other state. This is desired to ensure the agent always has enough ammunition when it needed to fire. A qualitative examination of the best-performing Transformer-based agent showed that the agent retained this behavior. An agent trained from scratch, on the other hand, would be unlikely to learn this behavior since it is not an optimal step to obtaining the maximum reward.

Hyperparameter	Values	Pearson	Spearman
Network Architecture			
Context length	1 – 20	-0.20	-0.09
Attention heads	2 – 8	-0.15	-0.15
Transformer layers	4 – 8	-0.29	-0.18
π and v layers	1 – 4	0.12	-0.08
(Pretraining) Behavior-Cloning			
Batch size	128 – 1024	-0.32	-0.07
Learning rate	$1^{-6} - 1^{-3}$	0.04	0.11
(Fine-tuning) PPO			
Batch size	128 – 1024	0.16	0.11
Learning rate	$1^{-6} - 1^{-3}$	-0.10	-0.17
Batches per update	4 – 100	-0.17	-0.26
Number of updates	3 – 10	-0.25	-0.27
Training steps	0.1M – 2M	-0.06	0.31
Curiosity	$1^{-3} - 1^{-1}$	-0.14	0.08
RIDE Environment			
Episode length	1k – 5k	0.28	0.45
Process interval	1 – 10	-0.70	-0.78

Table 3: Hyperparameter values and correlation coefficients from hyperparameter tuning Transformer+BC in our experiments

Hyperparameter searching is conducted over the 50 runs of our models (*MLP*, *MLP+BC*, *MLP+BC+FS*, and *Transformer+BC*) and to determine the extent to which each hyperparameter affects agents reward scores. Table 3 shows the correlation coefficients of each hyperparameter for training a *Transformer+BC* model, which we consider the training method with the best performance. The analysis of hyperparameter correlations with performance reveals that the duration of episodes and intervals between processes in the simulation environment have the greatest influence. Episode length affects training due to controlling if the model is given enough time to explore the problem space and accurately evaluate the model's effectiveness. Process interval controls the decision frequency of the agent. Smaller process intervals may result in actions being terminated prematurely which could negatively impact performance. The number of layers in the Transformer architecture and the number of attention heads also have a notable impact on model performance. PPO fine-tuning is impacted moderately by the number of batches included in each gradient update (batches per update) and number of updates. During pre-training, the batch size and learning rate utilized in behavior cloning did not have a large impact on performance. We attribute these last correlations to the relatively limited size of the datasets that the training processes were given. Correlations for the other methods can be found in the appendix.

Method	Accuracy	σ
MLP + BC	95.50	0.41
MLP + BC + FS	99.28	0.48
Transformer + BC	**99.60**	**0.25**

Table 4: Mean accuracy of each method on unseen test set of FSM trajectories after pretraining phase

Table 4 shows the mean accuracies of each method after the pretraining phase and before fine-tuning with PPO. Each method is evaluated on an unseen test set of trajectories (state and action pairs) produced by the FSM, and the model is measured on how often it predicts the correct action given the state. All three methods achieve high accuracy above 95%, however, *MLP+BC* is significantly lower than the other methods. It has 95.5% accuracy compared to *MLP+BC* and *Transformer+BC* which both obtain greater than 99% accuracy. This is likely due to lacking observations from previous time steps in order to make a decision (i.e., no frame stacking). With only the current observation, it is more difficult to encode a plan of what to do next. For example, say the agent reaches a wall, then takes a few steps back. An agent with a history of the past observation would remember that it had seen a wall and taken steps back, so it could then infer that the next action should be to turn away from the wall. An agent with only the current observation will not remember the past, and thus it is more likely to make a mistake, such as running back into the wall. This phenomenon may also account for the degenerate outliers in *MLP+BC* that give poor reward scores.

Conclusion

This paper showcases the effectiveness of behavior cloning approaches in simulation or gaming environments where state-of-the-art methods like PPO struggle to produce agents with acceptable task performance. Behavior cloning offers game designers and other users a straightforward method for taking a limited behavior that operates sub-optimally or only in a small part of the environment, and expanding it to cover a more complex environment while retaining important aspects of the user's original behavior. Additionally, behavior cloning approaches demonstrate stability during training, outperforming PPO baselines. Practitioners should consider using these methods in environments that are relatively stable and do not undergo significant changes over time.

Future extensions of this work include exploring interleaved offline and online stages of training, which could mitigate PPO's potential to forget behaviors learned via behavior cloning. Additionally, since FSM methods can be labor-intensive to author, alternative approaches such as providing natural language instructions that can be converted to code using code generation methods (Le, Chen, and Babar 2020; Lehman et al. 2022) could be investigated for expressing desired behavior.

Acknowledgments

The research reported in this document/presentation was performed in connection with contract number W912CG-20-P-0006 with the U.S. Army Contracting Command - Aberdeen Proving Ground (ACC-APG). The views and conclusions contained in this document/presentation are those of the authors and should not be interpreted as presenting the official policies or position, either expressed or implied, of ACC-APG, CCDC-SC or the U.S. Government unless so designated by other authorized documents. Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1.

Andreae, J. H. 1963. Stella: A scheme for a learning machine. In *In Proceedings of the 2nd IFAC Congress, Basle*, 497–502.

Army, H. D. o. t. 2017. *Tc 7-100.2 Opposing Force Tactics: December 2011*. CreateSpace Independent Publishing Platform.

Arnab, A.; Dehghani, M.; Heigold, G.; Sun, C.; Lučić, M.; and Schmid, C. 2021. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, 6836–6846.

Baker, B.; Akkaya, I.; Zhokhov, P.; Huizinga, J.; Tang, J.; Ecoffet, A.; Houghton, B.; Sampedro, R.; and Clune, J. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *arXiv preprint arXiv:2206.11795*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton: Princeton University Press.

Biewald, L. 2020. Experiment tracking with weights and biases. Software available from wandb.com.

Bonasso, R. P. 1988. What ai can do for battle management: A report of the first aaai workshop on ai applications to battle management. *AI Magazine* 9(3):77–77.

Boron, J., and Darken, C. 2020. Developing combat behavior through reinforcement learning in wargames and simulations. In *2020 IEEE Conference on Games (CoG)*, 728–731. IEEE.

Broyles, C.; Frei, N.; Botten, T.; Krywiski, J.; Kastenholz, T.; and Sanders, B. 2022. Developing our soldiers to outthink, outmaneuver, and outfight the enemy. *infantry*.

Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; and Mordatch, I. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems* 34:15084–15097.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

François-Lavet, V.; Henderson, P.; Islam, R.; Bellemare, M. G.; and Pineau, J. 2018. An introduction to deep reinforcement learning. *CoRR* abs/1811.12560.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Fujimoto, S.; Meger, D.; and Precup, D. 2018. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*.

Goswami, K.; Howley, E.; and Mannion, P. 2019. *Decision Making for Autonomous Car Driving using Deep Reinforcement Learning(DRL)*. Ph.D. Dissertation, National University Of Ireland, Galway.

Hartholt, A.; McCullough, K.; Fast, E.; Reilly, A.; Leeds, A.; Mozgai, S.; Ustun, V.; and Gordon, A. 2021. Introducing ride: Lowering the barrier of entry to simulation and training through the rapid integration & development environment. In *Proceedings of the 2021 Virtual Simulation Innovation Workshop*.

Hassaine, F.; Abdellaoui, N.; Yavas, A.; Hubbard, P.; and Vallerand, A. L. 2006. Effectiveness of jsaf as an open architecture, open source synthetic environment in defense experimentation. Technical report, Defence Research and Development Canada (DRDC) Ottawa.

Hastie, T.; Tibshirani, R.; Friedman, J. H.; and Friedman, J. H. 2009. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.

Hauser, K. 2021. Hyperparameter tuning for reinforcement learning with bandits and off-policy sampling. Master's thesis, Case Western Reserve University.

He, K.; Chen, X.; Xie, S.; Li, Y.; Dollár, P.; and Girshick, R. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 16000–16009.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2018. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence* 32(1).

Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. *Advances in neural information processing systems* 29.

Howard, R. 1960. *Dynamic Programming and Markov Processes*. Cambridge, MA: MIT Press.

Janner, M.; Li, Q.; and Levine, S. 2021. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems* 34:1273–1286.

Le, T. H. M.; Chen, H.; and Babar, M. A. 2020. Deep learning for source code modeling and generation: Models, applications and challenges. *CoRR* abs/2002.05442.

Lehman, J.; Gordon, J.; Jain, S.; Ndousse, K.; Yeh, C.; and Stanley, K. O. 2022. Evolution through large models.

Li, Y. 2017. Deep reinforcement learning: An overview. *CoRR* abs/1701.07274.

Li, Y. 2018. Deep reinforcement learning. *CoRR* abs/1810.06339.

Micheli, V.; Alonso, E.; and Fleuret, F. 2022. Transformers are sample efficient world models. *arXiv preprint arXiv:2209.00588*.

Michie, D. 1963. Experiments on the mechanisation of game learning. *Computer Journal* 1:232–263.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature* 518(7540):529–533.

Močkus, J. 1975. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference: Novosibirsk, July 1–7, 1974*, 400–404. Springer.

Paine, T. L.; Paduraru, C.; Michi, A.; Gulcehre, C.; Zolna, K.; Novikov, A.; Wang, Z.; and de Freitas, N. 2020. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055*.

Paulus, R.; Xiong, C.; and Socher, R. 2018. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

Pomerleau, D. A. 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural computation* 3(1):88–97.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1(8):9.

Ragan, E. D.; Bowman, D. A.; Kopper, R.; Stinson, C.; Scerbo, S.; and McMahan, R. P. 2015. Effects of field of view and visual complexity on virtual reality training effectiveness for a visual scanning task. *IEEE transactions on visualization and computer graphics* 21(7):794–807.

Rajeswaran, A.; Kumar, V.; Gupta, A.; Vezzani, G.; Schulman, J.; Todorov, E.; and Levine, S. 2017. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

Reed, S.; Zolna, K.; Parisotto, E.; Colmenarejo, S. G.; Novikov, A.; Barth-Maron, G.; Gimenez, M.; Sulsky, Y.; Kay, J.; Springenberg, J. T.; et al. 2022. A generalist agent. *arXiv preprint arXiv:2205.06175*.

Roland, A.; Shiman, P.; et al. 2002. *Strategic computing: DARPA and the quest for machine intelligence, 1983-1993*. MIT Press.

Ross, S., and Bagnell, D. 2010. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 661–668. JMLR Workshop and Conference Proceedings.

Saad, D. 1999. *On-line learning in neural networks*. Cambridge University Press.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Seymour, N. E.; Gallagher, A. G.; Roman, S. A.; O'brien, M. K.; Bansal, V. K.; Andersen, D. K.; and Satava, R. M. 2002. Virtual reality training improves operating room performance: results of a randomized, double-blinded study. *Annals of surgery* 236(4):458.

Singireddy, S.; Jha, S. K.; and Velasquez, A. 2023. Automaton distillation: A neuro-symbolic transfer learning approach for deep RL.

Sprague, N. 2015. Parameter selection for the deep q-learning algorithm. In *Proceedings of the Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 24.

Stefik, M. 1985. Strategic computing at darpa: Overview and assessment. *Communications of the ACM* 28(7):690–704.

Stein, E., and Kobrick, J. 1984. A brief history of the use of simulation techniques in training and performance assessment. Technical report, United States Army Research Institute of Environmental Medicine (USARIEM).

Stiennon, N.; Ouyang, L.; Wu, J.; Ziegler, D.; Lowe, R.; Voss, C.; Radford, A.; Amodei, D.; and Christiano, P. F. 2020. Learning to summarize with human feedback. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 3008–3021. Curran Associates, Inc.

Turnitsa, C.; Blais, C.; and Tolk, A. 2022. *Simulation and wargaming*. Wiley Online Library.

Uludağ, G.; Kiraz, B.; Etaner-Uyar, A. Ş.; and Özcan, E. 2013. A hybrid multi-population framework for dynamic environments combining online and offline learning. *Soft Computing* 17:2327–2348.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems* 30.

Wang, Z.; Zhang, K.; Zhang, J.; Chen, G.; Ma, X.; Xin, G.; Kang, J.; Zhao, H.; and Yang, Y. 2022. Deep reinforcement learning and adaptive policy transfer for generalizable well control optimization. *Journal of Petroleum Science and Engineering* 217:110868.

White, G. B. 1986. Artificial intelligence concepts and the war gaming environment: A case study using the tempo war game. Technical report, Air Force Inst of Tech Wright-Patterson AFB OH School of Engineering.

Whitmer, D. E.; Ullman, D.; and Johnson, C. I. 2019. Virtual reality training improves real-world performance on a speeded task. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 63, 1218–1222. SAGE Publications Sage CA: Los Angeles, CA.

Widrow, B.; Gupta, N. K.; and Maitra, S. 1973. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics* 3:455–465.

Xing, J.; Nagata, T.; Chen, K.; Zou, X.; Neftci, E.; and Krichmar, J. L. 2021. Domain adaptation in reinforcement learning via latent unified state representation. *CoRR* abs/2102.05714.

Xiong, Z.; Liu, X.; Zhong, S.; Yang, H.; and Walid, A. 2018. Practical deep reinforcement learning approach for stock trading. *CoRR* abs/1811.07522.

Zhang, B.; Rajan, R.; Pineda, L.; Lambert, N. O.; Biedenkapp, A.; Chua, K.; Hutter, F.; and Calandra, R. 2021. On the importance of hyperparameter optimization for model-based reinforcement learning. *CoRR* abs/2102.13651.

Hyperparameter Correlations

Correlations between hyperparameters and reward score for non-Transformer methods are shown in Tables 5 - 7.

Hyperparameter	**Pearson**	**Spearman**
Network Architecture		
Base layers	-0.22	-0.38
Hidden size	-0.17	-0.13
π and v layers	0.12	0.14
PPO		
Curiosity	-0.03	-0.22
Learning rate	0.03	0.31
Batch size	-0.12	-0.14
Batches per update	-0.24	-0.31
Number of updates	-0.10	-0.04
Training steps	0.45	0.66
RIDE Environment		
Process interval	0.43	0.30
Episode length	-0.03	-0.18

Table 5: MLP correlations

Hyperparameter	**Pearson**	**Spearman**
Network Architecture		
Base layers	0.18	-0.06
Hidden size	-0.03	-0.09
π and v layers	-0.14	-0.06
(Pretraining) Behavior-Cloning		
Batch size	-0.10	0.03
Learning rate	0.12	-0.03
(Fine-tuning) PPO		
Curiosity	-0.08	-0.22
Learning rate	-0.12	-0.33
Batch size	-0.09	-0.16
Batches per update	-0.11	-0.19
Number of updates	0.09	-0.00
Training steps	0.01	0.15
RIDE Environment		
Process interval	0.39	0.10
Episode length	-0.09	0.09

Table 6: MLP+BC correlations

Hyperparameter	**Pearson**	**Spearman**
Network Architecture		
Base layers	-0.15	-0.19
Hidden size	0.02	-0.11
π and v layers	-0.05	-0.04
Frames to stack	0.31	0.11
(Pretraining) Behavior-Cloning		
Batch size	0.05	-0.03
Learning rate	-0.25	-0.19
(Fine-tuning) PPO		
Curiosity	-0.16	-0.22
Learning rate	-0.30	-0.51
Batch size	0.13	0.09
Batches per update	-0.03	-0.21
Number of updates	-0.12	-0.13
Training steps	0.23	0.20
RIDE Environment		
Process interval	-0.05	-0.41
Episode length	0.03	0.01

Table 7: MLP+BC+FS correlations