Multi-Game Decision Transformers

Kuang-Huei Lee* **Ofir Nachum*** **Mengjiao Yang** **Lisa Lee**

Daniel Freeman **Winnie Xu** **Sergio Guadarrama** **Ian Fischer**

Eric Jang **Henryk Michalewski** **Igor Mordatch***

Google Research

Abstract

A longstanding goal of the field of AI is a method for learning a highly capable, generalist agent from diverse experience. In the subfields of vision and language, this was largely achieved by scaling up transformer-based models and training them on large, diverse datasets. Motivated by this progress, we investigate whether the same strategy can be used to produce generalist reinforcement learning agents. Specifically, we show that a single transformer-based model – with a single set of weights – trained purely offline can play a suite of up to 46 Atari games simultaneously at close-to-human performance. When trained and evaluated appropriately, we find that the same trends observed in language and vision hold, including scaling of performance with model size and rapid adaptation to new games via fine-tuning. We compare several approaches in this multi-game setting, such as online and offline RL methods and behavioral cloning, and find that our Multi-Game Decision Transformer models offer the best scalability and performance. We release the pre-trained models and code to encourage further research in this direction.[1]

1 Introduction

Building large-scale generalist models that solve many tasks by training on massive task-agnostic datasets has emerged as a dominant approach in natural language processing [18, 12], computer vision [19, 6], and their intersection [61, 4]. These models can adapt to new tasks (such as translation [63, 78]), make use of unrelated data (such as using high-resource language to improve translations of low-resource languages [17]), or even incorporate new modalities by projecting images into language space [46, 75]. The success of these methods largely derives from a combination of scalable model architectures [77], an abundance of unlabeled task-agnostic data, and continuous improvements in high performance computing infrastructure. Crucially, scaling laws [38, 31] indicate that performance gains due to scale have not yet reached a saturation point.

In this work, we argue that a similar progression is possible in the field of reinforcement learning, and take initial steps toward scalable methods that produce highly capable generalist agents. In contrast to vision and language domains, reinforcement learning has seen advocacy for the use of smaller models [16, 49, 8] and is usually either used to solve single tasks, or multiple tasks within the same environment. Importantly, training across multiple environments – with very different dynamics, rewards, visuals, and agent embodiments – has been studied less significantly.

*Equal contribution. [leekh, ofirnachum, imordatch]@google.com

[1]Additional information, videos and code can be seen at sites.google.com/view/multi-game-transformers.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

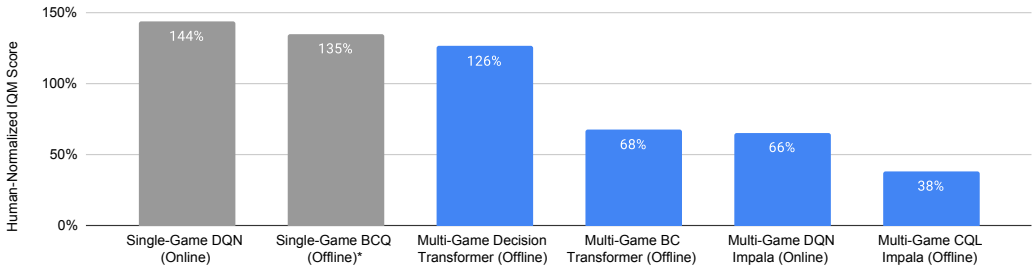


Figure 1: Aggregates of human-normalized scores (Inter-Quartile Mean) across 41 Atari games. Grey bars are single-game specialist models while blue are generalists. Single-game BCQ [21] results are from Gulcehre et al. [25]. Multi-game models are all trained on a dataset [1] with inter-quartile mean human-normalized score of 101%, which Multi-Game DT notably exceeds.

Specifically, we investigate whether a single model – with a single set of parameters – can be trained to act in multiple environments from large amounts of expert and non-expert experience. We consider training on a suite of 41 Atari games [9, 25] for their diversity, informally asking "Can models learn something universal from playing many video games?". To train this model, we use only the previously-collected trajectories from Agarwal et al. [1], but we evaluate our agent interactively. We are not striving for mastery or efficiency that game-specific agents can offer, as we believe we are still in early stages of this research agenda. Rather, we investigate whether the same trends observed in language and vision hold for large-scale generalist reinforcement learning agents.

We find that we can train a single agent that achieves 126% of human-level performance simultaneously across all games after training on offline expert and non-expert datasets (see Figure 1). Furthermore, we see similar trends that mirror those observed in language and vision: rapid fine-tuning to never-before-seen games with very little data (Section 4.5), a scaling relationship between performance and model size (Section 4.4), and faster training progress for larger models (Appendix G).

Notably, not all existing approaches to multi-environment training work well. We investigate several approaches, including treating the problem as offline decision transformer-based sequence modeling [14, 35], online RL [53], offline temporal difference methods [42], contrastive representations [56], and behavior cloning [60]. We find that decision transformer based models offer the best performance and scaling properties in the multi-environment regime. However, to permit training on both expert and non-expert trajectories, we find it is necessary to use a guided generation technique from language modeling to generate expert-level actions, which is an important departure from standard decision transformers.

Our contributions are threefold: First, we show that it is possible to train a single high-performing generalist agent to act across multiple environments from offline data alone. Second, we show that scaling trends observed in language and vision hold. And third, we compare multiple approaches for achieving this goal, finding that decision transformers combined with guided generation perform the best. It is our hope this study can inspire further research in generalist agents. To aid this, we make our pre-trained models and code publicly available.

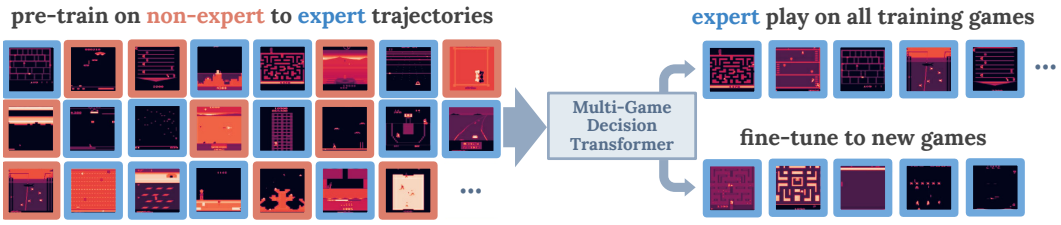


Figure 2: An overview of the training and evaluation setup. We observe expert-level game-play in the interactive setting after offline learning from trajectories ranging from beginner to expert.

2 Related Work

A generalist agent for solving a variety of environments has been a goal for artificial intelligence (AI) researchers since the inception of AI as a field of study [50]. This same reason motivated the introduction of the Atari suite (the Arcade Learning Environment, or ALE) as a testbed for learning algorithms [10]; in their own words, the ALE is for "empirically assessing agents designed for general competency." While the celebrated deep Q-learning [52] and actor critic [54] agents were among the first to use a single algorithm for all games, they nevertheless required separate training and hyperparameters for each game agent. Later works have demonstrated the ability to learn a single neural network agent on multiple Atari games simultaneously, either online [20] or via policy distillation [59, 67]. The aim of our work is similar – to learn a single agent for playing multiple Atari games – with a focus on offline learning. We demonstrate results with human-level competency on up to 46 games, which is unseen in the literature.

A closely related setting is learning to solve multiple tasks within the same or similar environments. For example in the robotics field, existing works propose to use language-conditioned tasks [48, 3, 34], while others posit goal-reaching as a way to learn general skills [51], among other proposals [37, 82]. In this work, we tackle the problem of learning to act in a large collection of environments with distinctively different dynamics, rewards, and agent embodiments. This complicated but important setting requires a different type of generalization that has been studied significantly less.

A concurrent work [65] also aims to train a transformer-based generalist agent based on offline data including for the ALE. This work differs from ours in that the offline training data is exclusively near-optimal and it requires prompting by expert trajectories at inference time. In contrast, we extend decision transformers [14] from the Upside-Down RL family [71, 68] to learn from a diverse dataset (expert and non-expert data), predict returns, and pick optimality-conditioned returns. Furthermore, we provide comparisons against existing behavioral cloning, online and offline RL methods, and contrastive representations [80, 56]. Other works that also consider LLM-like sequence modeling for a variety of single control tasks include [66, 84, 35, 23, 57].

3 Method

We consider a decision-making agent that at every time t receives an observation of the world $\mathbf{o}^t$, chooses an action a^t, and receives a scalar reward r^t. Our goal is to learn a single optimal policy distribution $P^*_\theta(a^t|\mathbf{o}^{\leq t}, a^{<t}, r^{<t})$ with parameters θ that maximizes the agent's total future return $R^t = \sum_{k>t} r^k$ on all the environments we consider.

3.1 Reinforcement Learning as Sequence Modeling

Following [14], we pose the problem of offline reinforcement learning as a sequence modeling problem where we model the probability of the next sequence token x_i conditioned on all tokens prior to it: $P_\theta(x_i|x_{<i})$, similar to contemporary decoder-only sequence models [12, 15, 62]. The sequences we consider have the form:

$$x \;=\; \langle ..., \mathbf{o}^t_1, ..., \mathbf{o}^t_M, \hat{R}^t, a^t, r^t, ... \rangle$$

where t represents a time-step, M is the number of image patches per observation (which we further discuss in Section 3.2), and $\hat{R}^t$ is the agent's target return for the rest of the sequence. Such a sequence order respects the causal structure of the environment decision process. Figure 3 presents an overview of our model architecture.

Returns, actions, and rewards are tokenized (See Section 3.2 for details), and we train the model to predict the next return, action, and reward discrete token in a sequence via standard cross-entropy loss. The sequence we consider is different from Chen et al. [14], which has $\langle ..., \hat{R}^t, \mathbf{o}^t, a^t, ... \rangle$. Our design allows predicting the return distribution and sampling from it, instead of relying on a user to manually select an expert-level return at inference time (See Section 3.4).

Predicting future value and rewards have been shown to be useful objectives for learning better representations in artificial reinforcement learning agents [47, 69, 44] and important signals for representation learning in humans [5]. Thus, while we may not directly use all of the predicted quantities, the task of predicting them encourages structure and representation learning of our

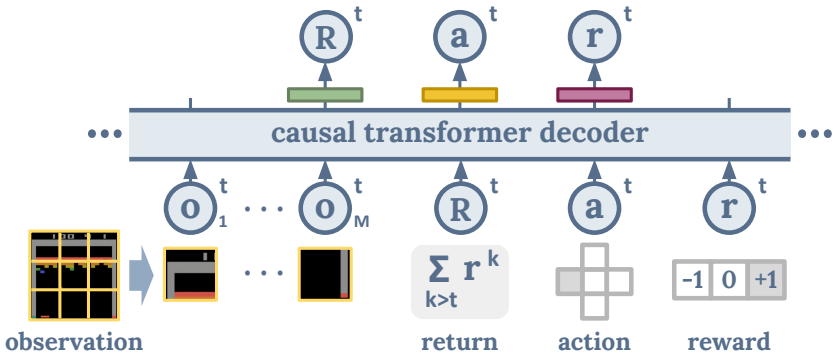


Figure 3: An overview of our decision transformer architecture.

environments. In this work, we do not attempt to predict future observations due to their non-discrete nature and the additional model capacity that would be required to generate images. However, building image-based forward prediction models of the environment has been shown to be a useful representation objective for RL [28, 27, 29]. We leave it for future investigation.

3.2 Tokenization

To generate returns, actions, and rewards via multinomial distributions similarly to language generation, we convert these quantities to discrete tokens. Actions a are already discrete quantities in the environments we consider. We convert scalar rewards to ternary quantities $\{-1, 0, +1\}$, and uniformly quantize returns into a discrete range shared by all our environments.[2]

Inspired by the simplicity and effectiveness of transformer architectures for processing images [19], we divide each observation image into a collection of M patches[3] (see Figure 3). Each patch is additively combined with a trainable position encoding and linearly projected into the input token embedding space. We experimented with using image tokenizations coming from a convolutional network, but did not find it to have a significant benefit and omitted it for simplicity.

We chose our tokenization scheme with simplicity in mind, but many other schemes are possible. While all our environments use a shared action space, varying action spaces when controlling different agent morphologies can still be tokenized using methods of [33, 43, 26]. And while we used uniform quantization to discretize continuous quantities, more sophisticated methods such as VQ-VAE [76] can be used to learn more effective discretizations.

3.3 Training Dataset

To train the model, we use an existing dataset of Atari trajectories (with quantized returns) introduced in [1]. The dataset contains trajectories collected from the training progress of a DQN agent [53]. Following [25], we select 46 games where DQN significantly outperforms a random agent. 41 games are used for training and 5 games are held out for out-of-distribution generalization experiments.

We chose 5 held-out games representing different categories including `Alien` and `MsPacman` (maze based), `Pong` (ball tracking), `SpaceInvaders` (shoot vertically), and `StarGunner` (shoot horizontally), to ensure out-of-distribution generalization can be evaluated on different types of games.

For each of 41 games, we use data from 2 training runs, each containing roll-outs from 50 policy checkpoints, in turn each containing 1 million environment steps. This totals 4.1 billion steps. Using the tokenization scheme in previous sections, the dataset contains almost 160 billion tokens.

As the dataset contains agent behaviors at all stages of learning, it contains both expert and non-expert behaviors. We do not perform any special filtering, curation, or balancing of the dataset. The

[2]The training datasets we use (Section 3.3) contains scalar reward values clipped to $[-1, 1]$. For return quantization, we use range $\{-20, ..., 100\}$ with bin size 1 in all our experiments as we find it covers most of the returns we observe in the datasets.

[3]We use 6x6 patches, where each patch corresponds to 14x14 pixels, in all our experiments.

motivation to train on such data instead of expert-only behaviors is twofold: Firstly, sub-optimal behaviors are more diverse than optimal behaviors and may still be useful for learning representations of the environment and consequences of poor decisions. Secondly, it may be difficult to create a single binary criteria for optimality as it is typically a graded quantity. Thus, instead of assuming only task-relevant expert behaviors, we train our model on all available behaviors, yet generate expert behavior at inference time as described in the next section.

3.4 Expert Action Inference

As described above, our training datasets contain a mix of expert and non-expert behaviors, thus directly generating actions from the model imitating the data is unlikely to consistently produce expert behavior (as we confirm in Section 4.7). Instead, we want to control action generation to consistently produce actions of highly-rewarding behavior. This mirrors the problem of discriminator-guided generation in language models, for which a variety of methods have been proposed [40, 79, 58].

We propose an inference-time method inspired by [40] and assume a binary classifier $P(\text{expert}^t|...)$ that identifies whether or not the behavior is expert-level before taking an action at time t. Following Bayes' rule, the distribution of expert-level returns at time t is then:

$$P(\text{expert}^t|R^t, ...) \propto \exp(\kappa(R^t - R_{low})/(R_{high} - R_{low}))$$

where R_{low} is the return lower bound and R_{high} is the return upper bound. Similarly to [70, 73, 74, 39], we define a binary classifier to be proportional to future return with inverse temperature κ[4]:

$$P(\text{expert}^t|R^t, ...) \equiv \exp(\kappa R^t)$$

This results in a simple auto-regressive procedure where we first sample high-but-plausible target returns R^t according to log-probability $\log P_\theta(R^t|...) + \kappa(R^t - R_{low})/(R_{high} - R_{low})$, and then sample actions according to $P_\theta(a^t|R^t, ...)$. See Figure 4 for an illustration of this procedure and Appendix B.3 for implementation details. It can be seen as a variation of return-conditioned policies [41, 71, 14] that automatically generates expert-level (but likely) returns at every timestep, instead of manually fixing them for the duration of the episode.

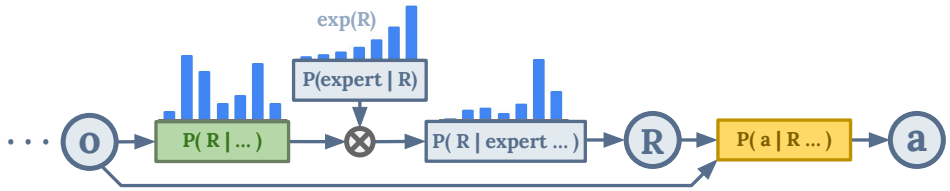


Figure 4: An illustration of our expert-level return and action sampling procedure. $P_\theta(R|...)$ and $P_\theta(a|R...)$ are the distributions learned by the sequence model.

Importantly, this formulation only affects the inference procedure of the model – training is entirely unaffected and can rely on standard next-token prediction frameworks and infrastructure. While we chose this formulation for its simplicity, controllable generation is an active area of study and we expect other more effective methods to be introduced in the future. As such, our contribution is to point out a connection between problems of controllable generation in language modeling and optimality conditioning in control.

4 Experiments

We formulate our experiments to answer a number of questions that are addressed in following sections:

- How do different **online and offline** methods perform in the multi-game regime?
- How do different methods **scale with model size**?
- How effective are different methods at **transfer to novel games**?

[4]We use $\kappa = 10$ in all our experiments.

- Does **multi-game decision transformer** improve upon training data?
- Does **expert action inference** (Section 3.4) improve upon behavioral cloning?
- Does training on **expert and non-expert data** bring benefits over expert-only training?

We also consider whether there are benefits to specifically using the **transformer architecture** in Appendix D, and qualitatively explore the attention behavior of these models in Appendix H.

4.1 Setup

Model Variants and Scaling. We base our decision transformer (DT) configuration on GPT-2 [12] as summarized in Appendix B.1. We report results for DT-200M (a Multi-Game DT with 200M parameters) if not specified otherwise. Other smaller variants are DT-40M and DT-10M. We set sequence length to 4 game frames for all experiments, which results in sequences of 156 tokens.

Training and Fine-tuning. We train all Multi-Game DT models on TPUv4 hardware and the Jaxline (Babuschkin et al. [7]) framework for 10M steps using the LAMB optimizer [81] with a $3 \cdot 10^{-4}$ learning rate, 4000 steps linear warm-up, no weight decay, gradient clip 1.0, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and batch size 2048. For fine-tuning on novel games, we train for 100k steps with a 10^{-4} learning rate, 10^{-2} weight decay and batch size of 256 instead. Both regimes used image augmentations as described in Appendix B.5.

Metrics. We measure performance on individual Atari games by human normalized scores (HNS) [53], i.e. $(\text{score} - \text{score}_{\text{random}})/(\text{score}_{\text{human}} - \text{score}_{\text{random}})$, or DQN-normalized scores, i.e. normalizing by the best DQN scores seen in the training dataset instead of using human scores. To create an aggregate comparison metric across all games, we use inter-quartile mean (IQM) of human-normalized scores across all games, following evaluation best practices proposed in [2]. Due to the prohibitively long training times, we only evaluated one training seed. We additionally report median aggregate metric in Appendix E.

4.2 Baseline Methods

BC Our Decision Transformer (Section 3.1) can be reduced to a transformer-based Behavioral Cloning (BC) [60] agent by removing the target return condition and return token prediction. Similar to what we do for Decision Transformer, we also learn BC models at different scales (10M, 40M, 200M parameters) while keeping other configurations unchanged.

C51 DQN As a point of comparison for online performance, we use the C51 algorithm [11] which is a variant of deep Q-learning (DQN) but with a categorical loss for minimizing the temporal difference (TD) errors. Following improvements suggested in Hessel et al. [30] as well as our own empirical observations, we use multi-step learning with $n = 4$. For the single-game experiments, we use the standard convolutional neural network (CNN) used in the implementation of C51 [13]. For the multi-game experiments, we modify the C51 implementation based on a hyperparameter search to use an Impala neural network architecture [20] with three blocks using 64, 128, and 128 channels respectively with a batch size of 128 and update period of 256.

CQL For an offline TD-based learning algorithm we use conservative Q-learning (CQL) [42]. Namely, we augment the categorical loss of C51 with a behavioral cloning loss minimizing $-\log \pi_Q(a|s)$, where (s, a) is a state-action pair sampled from the offline dataset and $\pi_Q(\cdot|s) = \mathrm{softmax}(Q(s, \cdot))$. Following the recommendations in Kumar et al. [42] we weight the contribution of the BC loss by 1 when using 100% of the offline data (multi-game training) and 4 when using 1% (single-game finetuning). For scaling experiments, we vary the number of blocks and channels in each block of the Impala: the number of blocks and channels is one of $(5 \text{ blocks}, 128 \text{ channels}) \approx 5\text{M params}$, $(10 \text{ blocks}, 256 \text{ channels}) \approx 30\text{M params}$, $(5 \text{ blocks}, 512 \text{ channels}) \approx 60\text{M params}$, $(10 \text{ blocks}, 512 \text{ channels}) \approx 120\text{M params}$.

CPC, BERT, and ACL For rapid adaptation to new games via fine-tuning, we consider representation learning baselines including contrastive predictive coding (CPC) [56], BERT pretraining [18], and attentive contrastive learning (ACL) [80]. All state representation networks are implemented

as additional multi-layer perceptrons (MLPs) or transformer layers on top of the Impala CNN used in C51 and CQL baselines. CPC uses two additional MLP layers with 512 units each interleaved with `ReLU` activation to represent $\phi(s)$, which is optimized by maximizing $\phi(s)^\top W \phi(s')$ of true transitions (s, s') and minimizing $\phi(s)^\top W \phi(\tilde{s})$ where $\tilde{s}$ is a state randomly sampled from the batch (including states from other games). For BERT pretraining, we use 2 self-attention layers with 4 attention heads of 256 units each and feed-forward dimension 512, and train $\phi(s)$ using BERT's masked self-prediction loss on a trajectory of sequence length 16. ACL shares the same model parametrization as BERT, with the inclusion of action prediction in the pretraining objective.

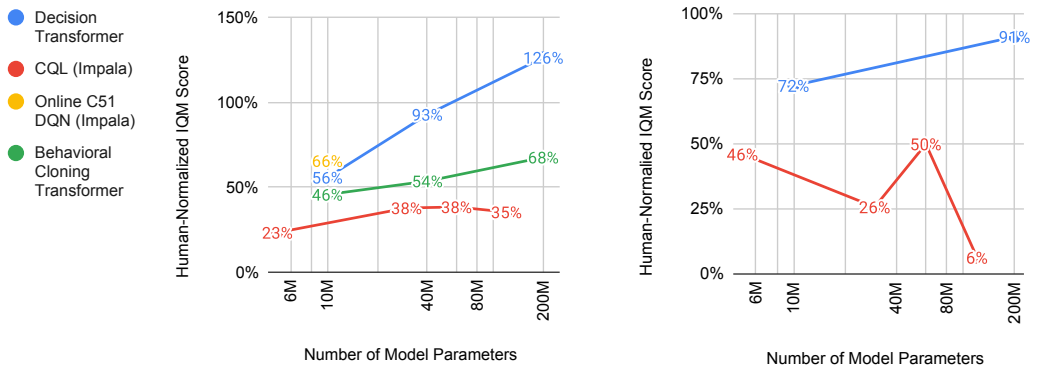
4.3 How do different online and offline methods perform in the multi-game regime?

We compare different online and offline algorithms in the multi-game regime and their single-game counterparts in Figure 1. We find that single-game specialists are still most performant. Among multi-game generalist models, our Multi-Game Decision Transformer model comes closest to specialist performance. Multi-game online RL with non-transformer models comes second, while we struggled to get good performance with offline non-transformer models. We note that our multi-game online C51 DQN median score of 68% (see Appendix E) which compares similarly to multi-game median Impala score of 70%, which we calculated from results reported by [20] for our suite of games.

We believe the apparent advantage of offline DT compared to online multi-game methods like C51 may be explained in part through classical differences between online and offline settings in RL [45]. Online methods must balance exploration with the ability to learn and generalize from experience, which could be challenging in the multi-game setting, whereas offline DT only needs to learn to distill and generalize from the fixed multi-game experience given to it (collected by specialist DQN agents [1]). Beyond the difference between online and offline, one could also argue that C51 suffers from more training instability than DT due to the use of a temporal difference (TD) loss, which we discuss in the next paragraph.

4.4 How do different methods scale with model size?

In large language and vision models, lowest-achievable training loss typically decreases predictably with increasing model size. Kaplan et al. [38] demonstrated an empirical scaling relationship between the capacity of a language model (NLP terminology for a next-token autoregressive generative model) and its performance (negative log likelihood on held-out data). These trends were verified over many orders of magnitude of model size, ranging from few-million parameter models to hundreds of billion parameter models.



(a) Scaling of IQM scores for all training games with different model sizes and architectures.

(b) Scaling of IQM scores for all novel games after fine-tuning DT and CQL.

Figure 5: How model performance scales with model size, on training set games and novel games. (Impala) indicates using the Impala CNN architecture.

We investigate whether similar trends hold for *interactive* in-game performance – not just training loss – and show a similar performance scaling trend in Figure 5a. Multi-Game Decision Transformer

performance reliably increases over more than an order of magnitude of parameter scaling, whereas the other methods either saturate, or have much slower performance growth.

In contrast, in Figures 5a and 5b, we find that CQL does not improve with increased model size, and actually shows a sharp drop in the performance of larger models on the fine-tuning tasks. Temporal Difference (TD) methods suffer greater instability with larger model size in the multi-game setting, leading to this "inverse" scaling. Indeed, our attempts at other objectives closer to pure TD (C51, DQN, DDQN) led to even worse results (which we do not report). We note that similar conclusions about instability with respect to network size have been made by other work [22].

We also find that larger models train faster, in the sense of reaching higher in-game performance after observing the same number of tokens. We discuss these results in Appendix G.

4.5 How effective are different methods at transfer to novel games?

Pretraining for rapid adaptation to new games has not been explored widely on Atari games despite being a natural and well-motivated task due to its relevance to how humans transfer knowledge to new games. Nachum and Yang [55] employed pretraining on large offline data and fine-tunining on small expert data for Atari and compared to a set of state representation learning objectives based on bisimulation [24, 83], but their pretraining and fine-tuning use the same game. We are instead interested in the *transfer* ability of pretrained agents to new games.

We hence devise our own evaluation setup by pretraining DT, CQL, CPC, BERT, and ACL on the full datasets of the 41 training games with 100M steps each, and fine-tuning one model per held-out game using 1% (1M steps) from each game. The 1% fine-tuning data is uniformly sampled from the 50M step dataset without quality filtering. DT and CQL use the same objective for pretraining and fine-tuning, whereas CPC, BERT, and ACL each use their own pretraining objective and are fine-tuned using the BC objective. All methods are fine-tuned for 100,000 steps, which is much shorter than training any agent from scratch. We additionally include training CQL from scratch on the 1% held-out data to highlight the benefit of rapid fine-tuning.

Fine-tuning performance on the held-out games is shown in Figure 6. Pretraining with the DT objective performs the best across all games. All methods with pretraining outperform training CQL from scratch, which verifies our hypothesis that pretraining on other games should indeed help with rapid learning of a new game. CPC and BERT underperform DT, suggesting that learning state representations alone is not sufficient for desirable transfer performance. While ACL adds an action prediction auxiliary loss to BERT, it showed little effect, suggesting that modeling the actions in the right way on the offline data is important for good transfer performance. Furthermore, we find that fine-tuning performance improves as the DT model becomes larger, while CQL fine-tuning performance is inconsistent with model size (see Figure 5b).

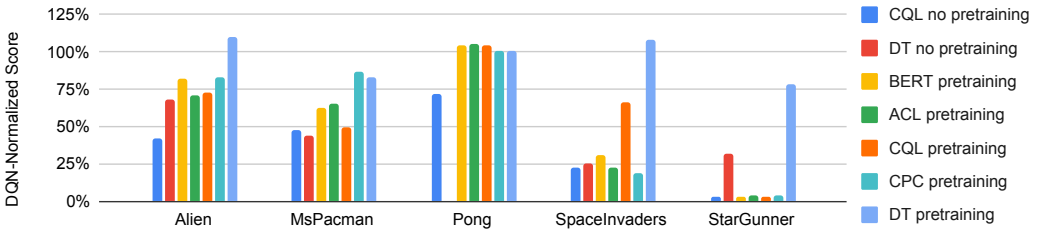


Figure 6: Fine-tuning performance on 1% of 5 held-out games' data after pretraining on other 41 games using DT, CQL, CPC, BERT, and ACL. All pretraining methods outperform training CQL from scratch on the 1% held-out data, highlighting the transfer benefit of pretraining on other games. DT performs the best among all methods considered.

4.6 Does multi-game decision transformer improve upon training data?

We want to evaluate whether decision transformer with expert action inference is capable of acting better than the best demonstrations seen during training. To do this, we look at the top 3 performing decision transformer model rollouts. We use top 3 rollouts instead of the mean across all rollouts

to more fairly compare to the *best* demonstration, rather than an average expert demonstration. We show percentage improvement over best demonstration score for individual games in Figure 7. We see significant improvement over the training data in a number of games.

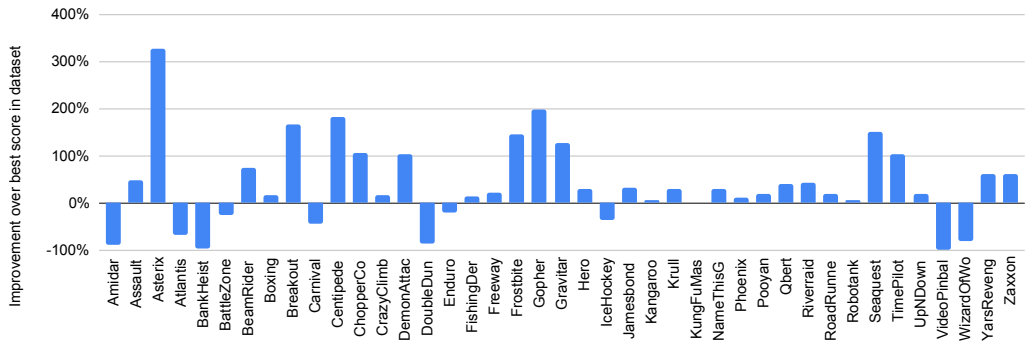


Figure 7: Percent of improvement of top 3 decision transformer rollouts over the best score in the training dataset. 0% indicates no improvement. Top-3 metric (instead of mean) is used to more fairly compare to the best – rather than expert average – demonstration score.

4.7 Does optimal action inference improve upon behavior cloning?

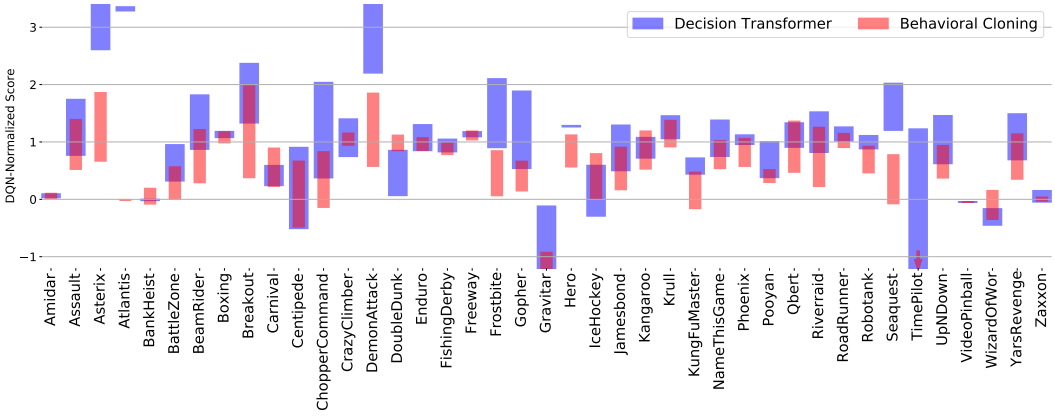


Figure 8: Comparison of per-game scores for decision transformer to behavioral cloning. Bars indicate $\pm$ standard deviation around the mean across 16 trials. We show DQN-normalized scores in this figure for better presentations.

In Figure 1 we see that IQM performance across all games is indeed significantly improved by generating optimality-conditioned actions. Figure 8 shows the mean and standard deviation of scores across all games. While behavior cloning may sometimes produce highly-rewarding episodes, it is less likely to do so. We find decision transformer outperforms behavioral cloning in 31 out of 41 games.

4.8 Does training on expert and non-expert data bring benefits over expert-only training?

We believe that, comparing to learning from expert demonstrations, learning from large, diverse datasets that include some expert data but primarily non-expert data help learning and improve performance. To verify this hypothesis, we filter our training data [1] from each game by episodic returns and only preserve top 10% trajectories to produce an expert dataset (see Appendix F for details). We use this expert dataset to train our multi-game decision transformer (DT-40M) and the transformer-based behavioral cloning model (BC-40M). Figure 9 compares these models trained on expert data and our DT-40M trained on all data.

We observe that (1) Training only on expert data improves behavioral cloning; (2) Training on full data, including expert and non-expert data, improves Decision Transformer; (3) Decision Transformer with full data outperforms behavioral cloning trained on expert data.

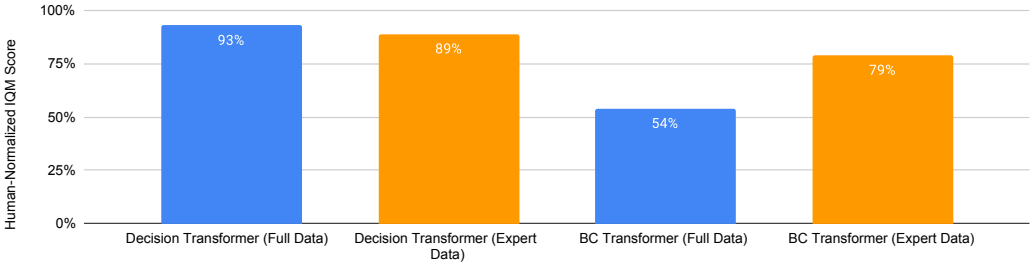


Figure 9: Comparison of 40M transformer models trained on full data and only expert data.

5 Conclusion

In the quest to develop highly capable and generalist agents, we have made important and measurable progress. Namely, our results exhibit a clear benefit of using large transformer-based models in multi-game domains, and the general trends in these results – performance improvements with larger models and the ability to rapidly fine-tune to new tasks – mirror the successes observed for large-scale vision and language models. Our results also highlight difficulties of online RL algorithms in handling the complexity of multi-game training on Atari. It is interesting to note that our best results are achieved by decision transformers, which essentially learn via supervised learning on sequence data, compared to alternative approaches such as temporal difference learning (more typical in reinforcement learning), policy gradients, and contrastive representation learning. This begs the question of whether online learning algorithms can be modified to be as "data-absorbent" as DT-like methods. While even our best generalist agents at times fall short of performance achieved by agents trained on a single task, this is broadly consistent with related works that have trained single models on many tasks [36, 65]. Still, our best generalist agents are already capable of outperforming the data they are trained on. We believe the trends suggest clear paths for future work – that, with larger models and larger suites of tasks, performance is likely to scale up commensurately.

Limitations. We acknowledge reasons for caution in over-generalizing our conclusions. Our results are based largely on performance in the Atari suite, where action and observation spaces are aligned across different games. It is unclear whether offline RL datasets such as Atari are of sufficient scale and diversity that we would see similar performance scaling as observed in NLP and vision benchmarks. Whether we can observe other forms of generalization, such as zero-shot adaptation, as well as whether our conclusions hold for other settings, remains unclear.

Societal Impacts. In the current setting, we do not foresee significant societal impact as the models are limited to playing simple video games. We emphasize that our current agents are not intended to interact with humans or be used outside of self-contained game-playing domains. One should exercise increased caution if extending our algorithms and methods to such situations in order to ensure any safety and ethical concerns are appropriately addressed. At the same time, the capability of decision making based on reward feedback – rather than purely imitation of the data – has the potential to be easier to align with human values and goals.

Acknowledgements

We would like to thank Oscar Ramirez, Roopali Vij, Sabela Ramos, Rishabh Agarwal, Shixiang (Shane) Gu, Aleksandra Faust, Noah Fiedel, Chelsea Finn, Sergey Levine, John Canny, Kimin Lee, Hao Liu, Ed Chi, and Luke Metz for their valuable contributions and support for this work.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[4] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

[5] William H Alexander and Samuel J Gershman. Representation learning with reward prediction errors. *arXiv preprint arXiv:2108.12402*, 2021.

[6] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6836–6846, 2021.

[7] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Tom Hennigan, Matteo Hessel, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Lena Martens, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/deepmind`.

[8] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

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

[10] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[11] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[13] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

[14] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[15] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[16] Giuseppe Cuccu, Julian Togelius, and Philippe Cudré-Mauroux. Playing atari with six neurons. *arXiv preprint arXiv:1806.01363*, 2018.

[17] Rishabh Dabral, Ganesh Ramakrishnan, Preethi Jyothi, et al. Rudder: A cross lingual video and text retrieval dataset. *arXiv preprint arXiv:2103.05457*, 2021.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[20] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[21] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[22] Hiroki Furuta, Tadashi Kozuno, Tatsuya Matsushima, Yutaka Matsuo, and Shixiang Shane Gu. Co-adaptation of algorithmic and implementational innovations in inference-based deep reinforcement learning. *Advances in neural information processing systems*, 34:9828–9842, 2021.

[23] Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. *arXiv preprint arXiv:2111.10364*, 2021.

[24] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR, 2019.

[25] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Thomas Paine, Sergio Gómez, Konrad Zolna, Rishabh Agarwal, Josh S Merel, Daniel J Mankowitz, Cosmin Paduraru, et al. Rl unplugged: A suite of benchmarks for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:7248–7259, 2020.

[26] Agrim Gupta, Silvio Savarese, Surya Ganguli, and Li Fei-Fei. Embodied intelligence via learning and evolution. *Nature communications*, 12(1):1–12, 2021.

[27] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

[28] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[29] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2020.

[30] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[31] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[32] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[33] Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *International Conference on Machine Learning*, pages 4455–4464. PMLR, 2020.

[34] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[35] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34, 2021.

[36] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[37] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[38] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[39] Hilbert J Kappen, Vicenç Gómez, and Manfred Opper. Optimal control as a graphical model inference problem. *Machine learning*, 87(2):159–182, 2012.

[40] Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. Gedi: Generative discriminator guided sequence generation. *arXiv preprint arXiv:2009.06367*, 2020.

[41] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

[42] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[43] Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

[44] Kuang-Huei Lee, Ian Fischer, Anthony Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in rl. *Advances in Neural Information Processing Systems*, 33: 11890–11901, 2020.

[45] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[46] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021.

[47] Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pages 1–9. PMLR, 2021.

[48] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[49] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[50] John McCarthy, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, 27(4):12–12, 2006.

[51] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34, 2021.

[52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[53] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[54] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[55] Ofir Nachum and Mengjiao Yang. Provable representation learning for imitation with contrastive fourier features. *Advances in Neural Information Processing Systems*, 34, 2021.

[56] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[57] Pedro A Ortega, Markus Kunesch, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Joel Veness, Jonas Buchli, Jonas Degrave, Bilal Piot, Julien Perolat, et al. Shaking the foundations: delusions in sequence models for interaction and control. *arXiv preprint arXiv:2110.10819*, 2021.

[58] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[59] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

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

[61] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[62] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[63] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[64] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? *Advances in Neural Information Processing Systems*, 34:12116–12128, 2021.

[65] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent, 2022. URL `https://arxiv.org/abs/2205.06175`.

[66] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[67] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[68] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

[69] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[70] Ross D Shachter. Probabilistic inference and influence diagrams. *Operations research*, 36(4):589–604, 1988.

[71] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

[72] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

[73] Emanuel Todorov. Linearly-solvable markov decision problems. *Advances in neural information processing systems*, 19, 2006.

[74] Marc Toussaint. Robot trajectory optimization using approximate inference. In *Proceedings of the 26th annual international conference on machine learning*, pages 1049–1056, 2009.

[75] Maria Tsimpoukelli, Jacob L Menick, Serkan Cabi, SM Eslami, Oriol Vinyals, and Felix Hill. Multimodal few-shot learning with frozen language models. *Advances in Neural Information Processing Systems*, 34: 200–212, 2021.

[76] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6309–6318, 2017.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[78] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, 2021.

[79] Kevin Yang and Dan Klein. Fudge: Controlled text generation with future discriminators. *arXiv preprint arXiv:2104.05218*, 2021.

[80] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. In *International Conference on Machine Learning*, pages 11784–11794. PMLR, 2021.

[81] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

[82] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[83] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[84] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. *arXiv preprint arXiv:2202.05607*, 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Contribution Statement

Kuang-Huei Lee Proposed project direction. Contributed to the JAX and TF Decision Transformer (DT) and BC code. Ran BC and DT experiments. Contributed to paper writing.

Ofir Nachum Proposed project direction. Contributed to TF code for DQN, CQL, and representation learning algorithms. Ran experiments for DQN and CQL. Contributed to paper writing.

Mengjiao Yang Contributed code in representation learning algorithms. Ran experiments for CPC, BERT, ACL pretraining + OOD finetuning. Contributed to paper writing.

Lisa Lee Contributed to TF codebase. Ran alternative environment experiments. Helped with paper editing.

Daniel Freeman Contributed to JAX training code and dataset generation pipelines. Generated datasets. Helped with paper writing.

Winnie Xu Contributed to JAX training infrastructure, beginner / expert dataset generation, augmentation and metrics pipelines. Ran experiments for alternative DT variants. Helped with paper writing. Work done while a Research Intern.

Sergio Guadarrama Helped with project direction and experiment discussions. Worked on paper editing.

Ian Fischer Helped with project direction and experiment discussions. Worked on paper editing.

Eric Jang Helped with project direction, contributed to building JAX infrastructure. Helped with paper writing.

Henryk Michalewski Contributed to building jaxline training and data processing infrastructure. Ran fine-tuning experiments. Helped with project direction and experiment discussions. Contributed to paper writing.

Igor Mordatch Proposed project direction. Contributed to building JAX training and data processing infrastructure. Ran DT experiments. Contributed to paper writing and visualizations.

B Implementation Details

B.1 Transformer network architecture

The input consists of a sequence of observations, returns, actions and rewards. Observations are images in the format $B \times T \times W \times H \times C$. We use 84×84 grayscale images (*i.e.*, $W = 84, H = 84, C = 1$). Similar to ViT [19], we extract M non-overlapping image patches, perform a linear projection and then rasterise them into d_{model}-dimensional 1D tokens. We define each patch to be 14×14 pixels (*i.e.*, $M = 6 \times 6 = 36$). A learned positional embedding is added to each of the patch tokens $\mathbf{o}_1, ..., \mathbf{o}_M$ to retain positional information as in ViT. As described in Section 3.2, returns are discretized into 120 buckets in $\{-20, ..., 100\}$, and rewards are converted to ternary quantities $\{-1, 0, +1\}$.

For the whole sequence $\langle ..., \mathbf{o}_1^t, ..., \mathbf{o}_M^t, \hat{R}^t, a^t, r^t, ... \rangle$, we learn another positional embedding at each position and add to each token embedding. We experimented with rotary position embedding [72], but did not find a significant benefit from them in our setting. On top of the token embeddings, our transformer models use a standard transformer decoder architecture.

A standard transformer implementation for sequence modeling would employ a sequential causal attention masking to prevent positions from attending to subsequent positions [77]. However, for the sequence $\langle ..., \mathbf{o}_1^t, ..., \mathbf{o}_M^t, \hat{R}^t, a^t, r^t, ... \rangle$ that we consider, we do not want to prevent the position corresponding to observation token $\mathbf{o}_m^t$ from accessing subsequent observation tokens $\{\mathbf{o}_{m'}^t : m' > m\}$ within the same timestep, since there is no clear sequential causal relation between image patches. Therefore, we change the sequential causal masking to allow observation tokens within the same timestep to access each other, but not subsequent positions after $\mathbf{o}_M^t$, *i.e.*$\hat{R}^t, a^t, r^t, \mathbf{o}_1^{t+1}, ..., \mathbf{o}_M^{t+1}, ...$

Table 1 summarizes the transformer configurations we use for each model size. We train these models on an internal cluster, each with 64 TPUv4. Due to prohibitively long training times, we only evaluated one training seed. For the 40M DT model, we perform four additional runs with different

seeds to investigate the sensitivity to random seeds. The mean and standard deviation of the median score across 5 random seeds are 0.79 and 0.06; the mean and standard deviation of IQM scores are 0.95 and 0.05. Compared to the differences to the baseline results and the results of other DT model sizes, we consider the variance to be relatively small.

Model	Layers	Hidden size D (d_{model})	Heads	Params	Training Time on 64 TPUv4
DT-10M	4	512	8	10M	1 day
DT-40M	6	768	12	40M	2 days
DT-200M	10	1280	20	200M	8 days

Table 1: Multi-Game Decision Transformer Variants

We performed hyperparameter search on DT-40M, and directly applied to DT-200M. With the set of hyperparameters working the best for smaller models, it still manifests a clear scaling trend of multi-game and transfer learning performance.

B.2 Fine-tuning protocol for Atari games

In the fine-tuning experiments, we reserved five games (Alien, MsPacman, Pong, Space Invaders and Star Gunner) to be used only for fine-tuning. These games were selected due to their varied gameplay characteristics. Each game was fine-tuned separately to measure the model's transfer performance for a fixed game. We use 1% of the original dataset (corresponding to roughly $500\,000$ transitions) to specifically test fine-tuning in low-data regimes.

B.3 Action and return sampling during in-game evaluation

We sample actions from the model with a temperature of 1. Inspired by Nucleus sampling (Holtzman et al. [32]), we only sample from the top 85th percentile action logits for all Decision Transformer models and Behavioral Cloning models (this parameter was selected to give highest performance for both models). While we train the model to predict actions for all timesteps in the sequence, during in-game evaluation, we execute the last predicted action in the sequence (conditioned on all past observations, and past generated actions, rewards, and target returns).

To generate target returns as discussed in Section 3.4, we sample them from the model with the temperature of 1 and the top 85th percentile logits. We use $\kappa = 10$ in all our experiments. To avoid storing the history of previously generated target returns (which may be difficult to incorporate into some RL frameworks), we experimented with autoregressively regenerating all target returns in the sequence, and found that to work well without requiring any special recurrent state maintenance outside of the model. Algorithm 1 has pseudocode for our expert return and action inference.

As an alternative way to generate expert-but-likely returns, we also experimented with simply generating N return samples from the model according to log-probability $\log P_\theta(R^t|...)$, and picking the highest one. We then generate the action conditioned on this largest picked return as before. This avoids needing the hyperparameter κ. In this setting, we found $N = 128$, inverse temperature of 0.75 for return sampling, no percentile cutoff for return sampling, and sampling from the top 50th percentile action logits with a temperature of 1 to work similarly well.

B.4 Evaluation protocol and Atari environment details

Our environment is the Atari 2600 Gym environment with pre-processing performed as in Agarwal et al. [1]. Our Atari observations are 84×84 grayscale images. We compress observation images to jpeg in the dataset (to keep dataset size small) and during in-game evaluation. All games use the same shared set of 18 discrete actions. For all methods, each game score is calculated by averaging over 16 model rollout episode trials. To reduce inter-trial variability, we do not use sticky actions during evaluation.

B.5 Image augmentation

All models were trained with image augmentations. We investigate training with the following augmentation methods: random cropping, random channel permutation, random pixel permutation,

Algorithm 1 Pseudocode for Expert Return and Action Inference (Section 3.4)

Given an environment E, the current time step t. $\kappa = 10$, the return upper bound $R_{high} = 100$, the return lower bound $R_{low} = -20$
Initialize a context window $\langle ..., r^{t-2}, \mathbf{o}_1^{t-1}, ..., \mathbf{o}_M^{t-1}, \hat{R}^{t-1}, a^{t-1}, r^{t-1}, \mathbf{o}_1^t, ..., \mathbf{o}_M^t \rangle$, abbreviated as $\ldots$ in the following.
 # Autoregressive return and action generation
while terminal state not reached yet **do**
  Compute 121 logits ($R^t = -20, \ldots, 100$) for the categorical return distribution $P(R^t|...)$
  # Increase logits proportionally to return magnitudes to prefer high magnitude
  Define $\log P(R^t|\text{expert}^t, ...) = \log P(R^t|...) + \kappa(R^t - R_{low})/(R_{high} - R_{low})$
  # Sample a return
  $R^t \sim P(R^t|\text{expert}^t, ...)$
  Compute logits for the categorical action distribution $P(a^t|R^t, ...)$
  # Sample an action
  $a^t \sim P(a^t|R^t, ...)$
  # Interact with the environment
  $\mathbf{o}_1^{t+1}, ..., \mathbf{o}_M^{t+1}, r^t \sim E_{step}(a^t)$
  $t = t + 1$ and shift the context window accordingly
end while

horizontal flip, vertical flip, and random rotations. We found random cropping and random rotations to work the best. (In our random cropping implementation, images of size 84×84 are padded on each side with 4 zero-value pixels, and then randomly cropped to 84×84.) In general, we aim to expand the domain of problems solved during training to similar kinds that we hope to generalize to by encoding useful inductive biases. We maintain the same random augmentation parameters for each window sequence. We apply data augmentation in both pre-training and fine-tuning.

C Baseline Implementation Details

BC Our BC model is effectively the same as our DT model but removing the return token $\hat{R}^t$ from the training sequence:

$$x \;=\; \langle ..., \mathbf{o}_1^t, ..., \mathbf{o}_M^t, a^t, r^t, ... \rangle$$

Instead of predicting a return token (distribution) given observation tokens $\mathbf{o}_1^t, ..., \mathbf{o}_M^t$ and the previous part of the sequence, we directly predict an action token (distribution), which also means that we remove return conditioning for the BC model. During evaluation, we sample actions with a temperature of 1, and sample from the top 85th percentile logits (as discussed in Appendix B.3). All other implementation details and configurations are identical to DT.

C51 DQN For single-game experiments, our implementation and training followed the details in [11] except for using multi-step learning with $n = 4$. For multi-game experiments we trained using the details provided in the main text; we ran the algorithm for 15M gradient steps ($\approx$ 4B environment steps $\approx$ 16B Atari frames).

CQL For CQL we use the same optimizer and learning rate as for C51 DQN. We use a per-replica batch size of 32 and run for 1M gradient steps on a TPU pod with 32 cores, yielding a global batch size of 256. During finetuning for each game, we copy the entire Q-network trained with CQL, and apply an additional 100k gradient steps of batch size 32 on a single CPU, where each batch is sampled exclusively from the offline dataset of the finetuned game. We also experimented with smaller learning rates (0.00003 instead of the default 0.00025) and larger batch sizes (1024, 4096) but found the results largely unchanged. We also tried using offline C51 and double DQN as opposed to CQL, and found performance to be worse.

CPC For the CPC baseline [56], we apply a contrastive loss between $\phi(o_t), \phi(o_{t+1})$ using the objective function

$$-\,\phi(o_{t+1})^\top W \phi(o_t) + \log \mathbb{E}_{\tilde{s}\sim\rho}[\exp\{\phi(\tilde{o})^\top W \phi(o_t)\}], \tag{1}$$

where W is a trainable matrix and ρ is a non-trainable prior distribution; for mini-batch training we set ρ to be the distribution of states in the mini-batch. The state representations $\phi(o)$ is parametrized by CNNs followed by two MLP layers with 512 units each interleaved with ReLU activation. For the CNN architecture, we used the C51 implementation with an Impala neural network architecture of three blocks using 16, 32, and 32 channels respectively, and trained with a batch size of 256 and learning rate of 0.00025 both during pretraining and downstream BC adaptation. We conduct representation learning for a total of 1M gradient steps, and finetune on 1% data for 100k steps every 50k steps of representation learning and report the best finetuning results.

BERT and ACL Our BERT and ACL baselines are based on the representation learning objectives described in [80]. For the BERT [18] state representation learning baseline, we (1) take a sub-trajectory $o_{t:t+k}, a_{t:t+k}, r_{t:t+k}$ from the dataset (without special tokenization as in DT), (2) randomly mask a subset of these, (3) pass the masked sequence into a transformer, and then (4) for each masked input state o_{t+i}, apply a contrastive loss between its representation $\phi(o_{t+i})$ and the transformer output $\text{Transformer}[i]$ at the corresponding sequence position:

$$-\phi(o_{t+i})^\top W \,\text{Transformer}[i] + \log \mathbb{E}_{\tilde{o}\sim\rho}[\exp\{\phi(\tilde{o})^\top W \,\text{Transformer}[i]\}], \tag{2}$$

where ρ is the distribution over states in the mini-batch. For attentive contrastive learning (ACL) [80], we apply an additional action prediction loss to the output of BERT at the sequence positions of the action inputs.

To parameterize ϕ, we use the same CNN architecture as in CPC, while the transformer is parameterized by two self-attention layers with 4 attention heads of 256 units each and feed-forward dimension 512. The transformer does not apply any additional directional masking to its inputs. We used $k = 16$.

Pretraining and finetuning is analogous to CPC. Namely, when finetuning we take the pretrained representation ϕ and use a BC objective for learning a neural network (two MLP layers with 512 units each) policy on top of this representation.

D Comparisons between transformers and convolution networks

Decision Transformer is an Upside-Down RL (UDRL) [68, 71] implementation that uses the transformer architecture and considers RL as a sequence modeling problem. To understand the benefit of the transformer architecture, we compare to a UDRL implementation that uses feed-forward, convolutional Impala networks [20]. Specifically, we use the same return, action, and reward tokenizers as in DT, and only replace the observation (four consecutive Atari frames stacked together) encoding to use the Impala architecture. Similar to what we do for CQL, we also experiment with different sizes of the Impala architecture by varying the number of blocks and channels in each block of the Impala network: the number of blocks and channels is one of $(5 \text{ blocks}, 128 \text{ channels}) \approx 5\text{M params}$, $(10 \text{ blocks}, 256 \text{ channels}) \approx 30\text{M params}$, $(5 \text{ blocks}, 512 \text{ channels}) \approx 60\text{M params}$. We use a $(768, 768)$ 2-layer fully-connected head to predict the next return token from observation embedding; another $(768, 768)$ head to predict the next action token from a concatenation of observation embedding and return token embedding; and another $(768, 768)$ head to predict the next reward token from a concatenation of observation embedding, return token embedding, and action token embedding.

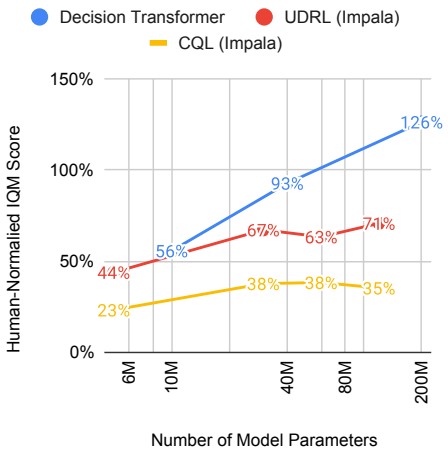


Figure 10: Performance scaling with model size for UDRL and CQL (Impala architecture) compared to Decision Transformer.

The input to the model is slightly different from what we have for DT: Instead of considering a T-timestep sub-trajectory ($T = 4$) where each timestep contains $\mathbf{o}^t, R^t, a^t, r^t$, we stack T image

frames (as common in [53]), and only consider R^t, a^t, r^t from the last timestep. All other design choices and evaluation protocols are the same as DT.

Figure 10 shows clear advantages of Decision Transformer over UDRL with the Impala architecture. In the comparison between UDRL (Impala) and CQL that uses the same Impala network at each model size we evaluated, we observe that UDRL (Impala) outperforms CQL. The results show that the benefits of our method come not only from using network architectures, but also from the UDRL formulation. While the reasons for the benefits of transformer architecture over other neural networks are still an open question in general (see e.g., [64]), our hypothesis is that transformers allow for easier discovery of correlations between components of the input and output, due to the fact that transformers process the input as a flat sequence with attention allowed between any patch, action, or return token. Although it is not feasible to compare transformer with all possible convolutional architectures due to the broad design space, we believe these empirical results still show a clear trend favoring both UDRL and transformer architectures.

E Comparisons between methods using median human normalized scores

We used inter-quartile mean (IQM) to aggregate performance over individual games in Figure 1. Median is another metric commonly used to aggregate scores (although it has issues as discussed in [2]: it has high variability, and in the most extreme case, the median is unaffected by zero performance on nearly half of the tasks.). For completeness, we report the median scores for all methods:

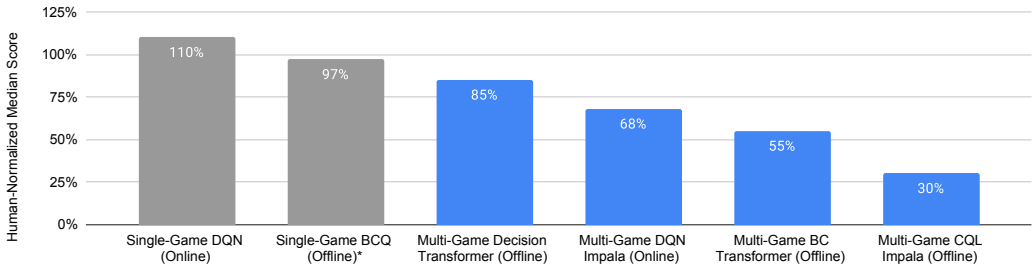


Figure 11: Median human-normalized score across 41 Atari games. Grey bars are single-game specialist models while blue are generalists. Single-game BCQ results are from Gulcehre et al. [25].

For expert-filtering experiments in Section 4.8, we also provide the plot of expert filtering effects with median human-normalized scores in Figure 12. We note that ranking of various configurations do not change across aggregate metrics.

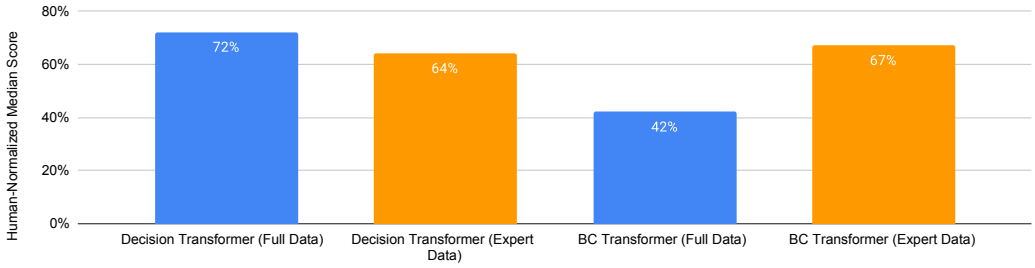


Figure 12: Median human-normalized scores of 40M transformer models trained on full data and only expert data.

For Upside-Down RL comparison experiments Appendix D, we also provide median human-normalized scores in Figure 13.

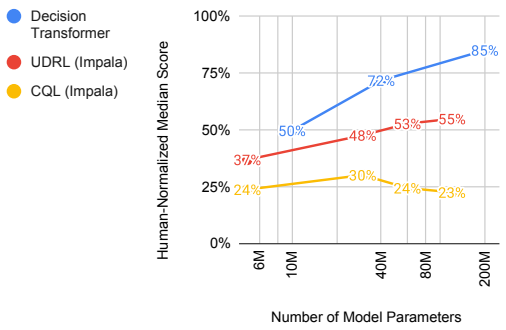


Figure 13: How UDRL (Impala architecture) median human-normalized score scales with model size on training set games, in comparisons with Decision Transformer and CQL (Impala architecture).

F Details of Expert Dataset Generation

To generate the expert dataset for experiments in Section 4.8, we we filter our training data [1] from each game by episodic returns and only preserve top 10% trajectories to produce an expert dataset. We plot of return histograms for reference in Figure 14.

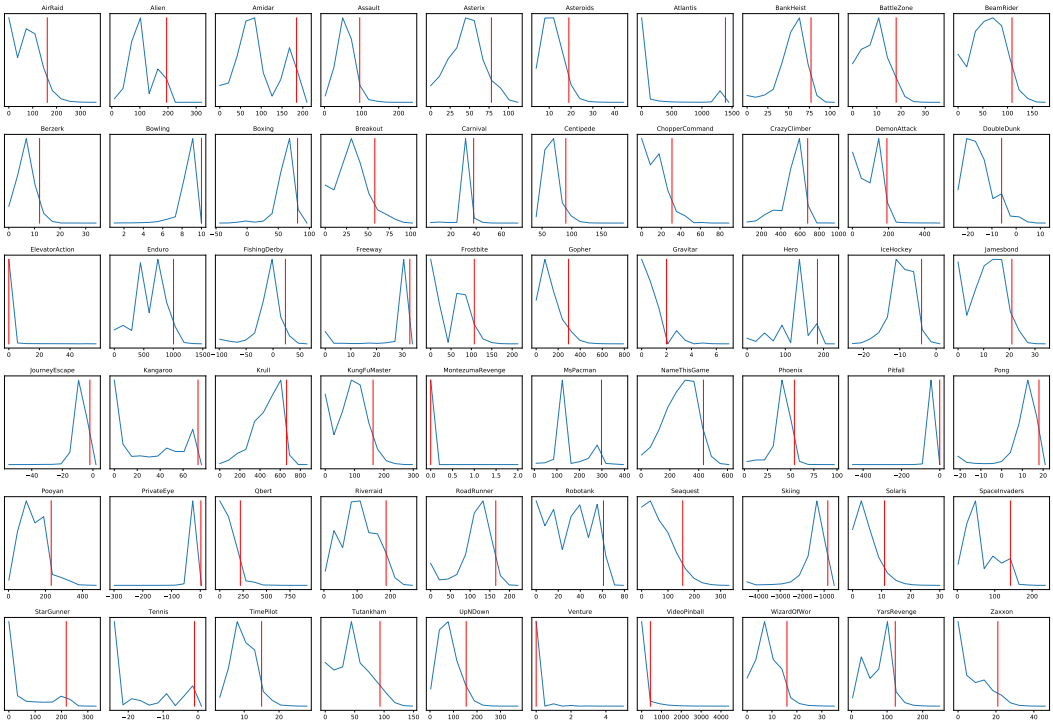


Figure 14: Histograms of rollout performance from [1] used to generate the expert dataset, with (unnormalized) score-density on the vertical axis, and game score (rewards are clipped) on the horizontal axis. We indicate the 90th percentile performance cutoff with a red vertical line for each game. Rollouts that exceeded this score threshold were included in the expert dataset.

G Effect of Model Size on Training Speed

It is believed that large transformer-based language models train faster than smaller models, in the sense that they reach higher performance after observing a similar number of tokens [38, 15]. We

find this trend to hold in our setting as well. Figure 15 shows an example of performance on two example games as multi-game training progresses. We see that larger models reach higher scores per number of training steps taken (thus tokens observed).

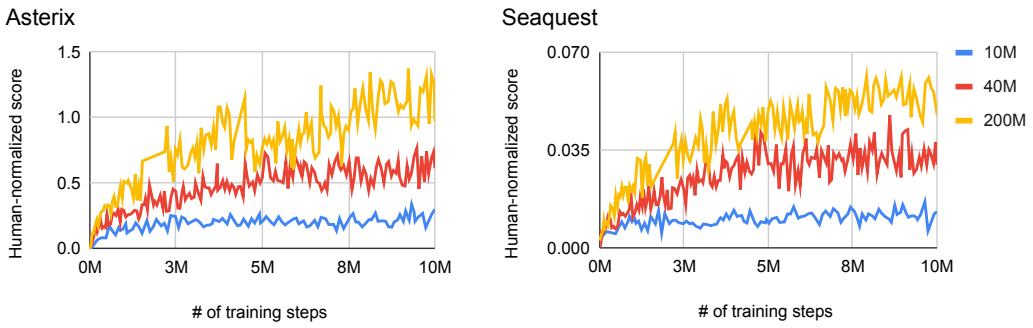
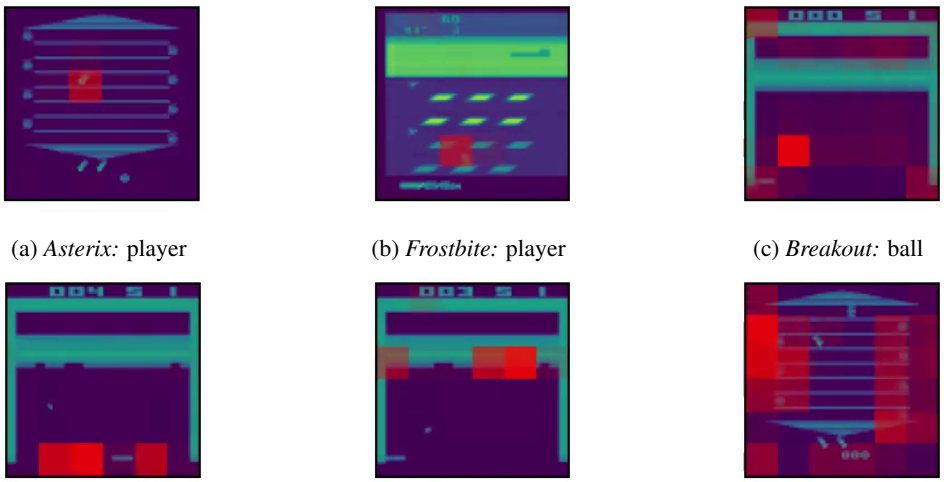


Figure 15: Example game scores for different model sizes as multi-game training progresses.

H Qualitative Attention Analysis

We find that the Decision Transformer model consistently attends to observation image patches that contain meaningful game entities. Figure 16 visualizes selected attention heads and layers for various games. We find heads consistently attend to entities such as player character, player's free movement space, non-player objects, and environment features.



(a) *Asterix:* player (b) *Frostbite:* player (c) *Breakout:* ball

(d) *Breakout:* no paddle (e) *Breakout:* unbroken blocks (f) *Asterix:* non-players

Figure 16: Example image patches attended (red) for predicting next action by Decision Transformer.

I Raw Atari Scores

We report full raw scores of 4 training Atari games for best performing sizes of multi-game models in Table 2.

Game Name	DT (200M)	BC (200M)	Online DQN (10M)	CQL (60M)
Amidar	101.5	101.0	629.8	4.0
Assault	2,385.9	1,872.1	1,338.7	820.1
Asterix	14,706.3	5,162.5	2,949.1	950.0
Atlantis	3,105,342.3	4,237.5	976,030.4	16,800.0
BankHeist	5.0	63.1	1,069.6	20.0
BattleZone	17,687.5	9,250.0	26,235.2	5,000.0
BeamRider	8,560.5	4,948.4	1,524.8	3,246.4
Boxing	95.1	90.9	68.3	100.0
Breakout	290.6	185.6	32.6	62.0
Carnival	2,213.8	2,986.9	2,021.2	440.0
Centipede	2,463.0	2,262.8	4,848.0	2,904.0
ChopperCommand	4,268.8	1,800.0	951.4	400.0
CrazyClimber	126,018.8	123,350.0	146,362.5	139,300.0
DemonAttack	23,768.4	7,870.6	446.8	1,202.0
DoubleDunk	-10.6	-1.5	-156.2	-2.0
Enduro	1,092.6	793.2	896.3	729.0
FishingDerby	11.8	5.6	-152.3	18.4
Freeway	30.4	29.8	30.6	32.0
Frostbite	2,435.6	782.5	2,748.4	408.0
Gopher	9,935.0	3,496.3	3,205.6	700.0
Gravitar	59.4	12.5	492.5	0.0
Hero	20,408.8	13,850.0	26,568.8	14,040.0
IceHockey	-10.1	-8.3	-10.4	-10.5
Jamesbond	700.0	431.3	264.6	500.0
Kangaroo	12,700.0	12,143.8	7,997.1	6,700.0
Krull	8,685.6	8,058.8	8,221.4	7,170.0
KungFuMaster	15,562.5	4,362.5	29,383.1	13,700.0
NameThisGame	9,056.9	7,241.9	6,548.8	3,700.0
Phoenix	5,295.6	4,326.9	3,932.5	1,880.0
Pooyan	2,859.1	1,677.2	4,000.0	330.0
Qbert	13,734.4	11,276.6	4,226.5	11,700.0
Riverraid	14,755.6	9,816.3	7,306.6	3,810.0
RoadRunner	54,568.8	49,118.8	25,233.0	50,900.0
Robotank	63.2	44.6	9.2	17.0
Seaquest	5,173.8	1,175.6	1,415.2	643.0
TimePilot	2,743.8	1,312.5	-883.1	2,400.0
UpNDown	16,291.3	10,454.4	8,167.6	5,610.0
VideoPinball	1,007.7	1,140.8	85,351.0	0.0
WizardOfWor	187.5	443.8	975.9	500.0
YarsRevenge	28,897.9	20,738.9	18,889.5	19,505.4
Zaxxon	275.0	50.0	-0.1	0.0

Table 2: Raw scores of 41 training Atari games for best performing multi-game models.