Decision Transformer With Tokenized Action Space

Graham Annett [1], **Tim Andersen** [1]

[1] Boise State University
grahamannett@u.boisestate.edu

Abstract

In the realm of reinforcement learning (RL), the vision of developing models capable of generalizing across myriad environments has remained elusive. While the Decision Transformer (DT) and Trajectory Transformer (TT) have made headway, challenges remain, especially in their adaptability to diverse problems without the need for extensive retraining. We introduce an innovative approach that emphasizes action tokenization, borrowing concepts from large language models (LLM). Rather than comprehensively tokenizing the state, action, and reward trajectories, we introduce an action-centric tokenization schema. This has the advantage of retaining the state space in its native form and fosters environment-agnostic model training. Two tokenized action embeddings, namely, ActionTokenizedEmbedding and ActionTokenizedSpreadEmbedding, are explored, providing flexibility and adaptability. Preliminary results show the model's potential for quick acclimation to new terrains.

Introduction

In the rapidly evolving domain of reinforcement learning (RL), the promise of generalization across diverse environments has been a long-standing goal. Recent strides in deep learning have illuminated a path forward with the rise of the Decision Transformer (DT) (Chen, Lu, Rajeswaran, Lee, Grover, Laskin, Abbeel, Srinivas, and Mordatch 2021) and Trajectory Transformer (TT) (Janner, Li, and Levine 2021) which vastly simplify many aspects of RL. These models, highly capable and achieving remarkable results for many tasks and environments, have allowed training of offline RL models that is in many ways more similar to the training of language models than to the training of traditional RL algorithms.

While the DT and TT models rely heavily on the transformer architecture, they are still relatively specific to the environment they are trained on and there has yet to be anything similar to the foundation models that have been developed for language models. Some aspects that prevent these RL models from being as adaptable as their language model counterparts is the need for model components that are environment or task specific which differs from the language models that can be trained on a single task and then fine-tuned for a new task with only a single output head or often no architecture change. Other issues include the need for quite large datasets as transformers are known to be sample inefficient, and the need to condition on future rewards which can fail in stochastic environments. This work addresses these issues in part by adapting the model to resemble language models more closely to allow advancement in their ability to work on downstream tasks.

The present study introduces a novel approach that leverages the strengths of the Decision Transformer, specifically focusing on the action space through a unique tokenization schema. By preserving the state space in its original form and adopting this action-only tokenization, our model is trained in an environment-agnostic fashion. This approach emphasizes the centrality of actions in the decision-making process, enabling swift adaptability in unfamiliar terrains. Our work sets the stage for the development of foundational models in RL, proposing a vision where a single training schema can be employed across tasks, with simple fine-tuning in both supervised and unsupervised manner sufficing for novel environments.

Background

Training an agent for a task or environment with only pre-collected data is known as offline reinforcement learning (Levine et al. 2020). These trajectories are typically composed of a sequence of states, actions, and rewards, but may also include additional information, are collected either by an online algorithm or from expert examples, with the goal that an offline agent can learn from these examples without having to interact with the environment. In comparison to the more studied and well known realm of online algorithms, offline RL is particularly appealing in scenarios where agents face costly or risky online data collection, and research focused on making offline agents more able to adept to new environments is a critical area of research.

The field of RL has many different ways in which it can be categorized but this work uses the framing of models or agents into two broad categories: model-based and model-free. Although each of these framings have a variety of approaches and techniques associated with them, the models that we focus on are model-free. Model-free methods are often seen as appealing due to their algorithmic simplicity

(causal sequence modeling) and architecture (a transformer based architecture). In terms of the models and approaches that guide this work, model-based approaches (i.e. the Trajectory Transformer (Janner, Li, and Levine 2021)) have attractive properties such as the ability to generate an arbitrary future sequence that can be used for bootstrapping (Wang et al. 2022) and beam-search. The drawback of this approach is that due to the need to tokenize across all the input modalities, the resulting sequences are exponentially larger than the original sequence. Often this sequence length growth can largely be attributed to the observation space, which is typically much larger than the action space. Overall this approach is conceptually the most similar to how many of the current state of the art language models work.

On the other hand, transformer based model-free approaches that frame the problem as a sequence modeling task focused only on predicting the next action conditioned on some reward(Chen et al. 2021), do not suffer from this exponential growth in sequence length, but are not without their own challenges such as brittleness in stochastic environments and the need for future rewards (Paster, McIlraith, and Ba 2022; Gao et al. 2023; Meng et al. 2022; Yamagata, Khalil, and Santos-Rodriguez 2022).

Both of these approaches (DT and TT) contain many of the same framings and aspects, for example the model is mostly comprised of a decoder transformer architecture with a causal masking scheme, and given a trajectory of (s_t, a_t, r_t), the inputs are eventually interleaved to create a sequence similar to (although the ordering and number per modality can be different):

$$(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T) \tag{1}$$

The relative computational efficiency and simplicity of the DT in particular, which embeds each of these modalities independently and results in a single vector of some embedding size per timestep, means that while the model can typically ingest longer contexts than a similar TT model, the trained model parameters are highly specific to a particular dataset and environment. Recent work aimed at devising general pretraining regimens (Sun et al. 2023) such that the model can be fine-tuned downstream for a different environment appear to work well, but depend on the state space being similar dimensionality between tasks (e.g. an RGB image).

Although it is possible to fine-tune pretrained models on downstream tasks for model sizes typically seen in offline RL (Tarasov et al. 2023), this is unlikely to be the case for large foundational models for RL. For many researchers, the usage of LLMs (Touvron et al. 2023) (Jiang et al. 2023) is impractical without access to substantial resources (e.g., 80GB A100/H100 GPUs) and technical know how (Zhao et al. 2023). Despite these concerns, techniques that allow researchers to use LLMs for RL tasks have been successful by primarily adapting techniques such as LoRA (Hu et al. 2021) and largely benefit from ideas originally put forth for NLP tasks (Shi et al. 2023).

The ability to adapt larger models initially trained on NLP tasks (Reid, Yamada, and Gu 2022) hints at the notion that large generalized RL models are indeed possible and could be used in a manner which does not require vast computational resources or large amounts of additional training data, but would likely require a more generalized approach to training which is the focus of this work.

Method

Our approach follows the basic formulation put forth in the original DT work (Chen et al. 2021), where an offline dataset $\mathcal{D}_{\text{offline}}$ is comprised of many trajectories $\tau = (s_T, a_T, r_T)$ that are tuples of states $s \in S$, actions $a \in A$ and rewards $r \in R$. This dataset is collected from expert examples or another algorithm typically in an online manner.

The goal of our model is not to maximize some expected reward (i.e. a typical approach for RL algorithms), instead the model is conditioned on some starting reward A which is computed from the discounted sum of returns-to-go, and given a trajectory τ the model is trained to minimize a loss calculated from the predicted actions at each timestep t. This can be represented as follows:

$$\mathcal{L}(\tau) = \sum_{t=0}^{T-1} P_\theta(a_t | s_t, a_{<t}, r_{<t}, A) \tag{2}$$

For our work this value can be computed from the predicted action at each timestep and is the standard $\mathcal{L}_2$ loss:

$$\mathcal{L}_t(\tau) = ||a_t - \hat{a}_t||^2 \tag{3}$$

where $\hat{a}_t$ is the predicted action at timestep t.

The reason for choosing to tokenize only actions is that an action space is generally smaller than the state space. This allows the spreading of the actions into the modality dimension while limiting the growth of the context length (as opposed to spreading the state vector which would grow the context length by a much larger factor). In addition to this reasoning, if one were to tokenize and spread all inputs (e.g. actions, states and rewards), the method becomes much more similar to that of TT (Janner, Li, and Levine 2021), which we found needed the works beam search method to generate actions that are of similar quality to the DT model. As our model outputs a single action vector and does not utilize a decoding schema (unless the action requires scaling beyond $(-1, 1)$), the beam search method is not applicable to our model.

Action Tokenization

The key idea of our approach is that we tokenize select types, but not all, of the input modalities (i.e. state, actions, rewards). By doing so we are able to train our model to be more adaptable to new environments as new environments will require fewer aspects of the model to be adjusted.

Tokenization Schema We tokenize the action space using a quantization schema, which transforms continuous values into a discrete set of tokens. To tokenize our actions we consider each feature of the action space independently and tokenize as such:

$$T_i = Q_i(a_i) + \sum_{j=0}^{i-1} |Q_j| \tag{4}$$

In equation 4, T_i is the token for the i-th feature, $Q_i(a_i)$ is the quantize function for the i-th feature applied to the action a_i, and $|Q_j|$ is the number of tokens for the j-th feature. While the sum runs over all preceding features, which means that the i-th feature can be represented by its own range of tokens, there is no particular reason why different features (i.e. a^i and a^{i+1}) cannot both be contained in the same token range (by omitting the offset $|Q_j|$). We discuss the implications and results of this choice further in the results section.

Tokenization Embedding We investigate two differing approaches for handling tokenized actions: **ActionTokenizedEmbedding** and **ActionTokenizedSpreadEmbedding**. In the former, actions are pooled per timestep and then interleaved with states and rewards. In the latter, actions are unpacked along the modality dimension (e.g. along the dimension the states and rewards are initially stacked upon), allowing for an embedding layer that utilizes the same parameters across different tasks and environments.

By distributing actions across the modality dimension, the **ActionTokenizedSpreadEmbedding** approach enables the model to generate the same number of actions that the unfolded action features have been spread to in an autoregressive manner. This flexibility allows us to either generate a predefined number of actions by padding the input or to dynamically produce the required number of actions in the same way LLMs sequentially generate tokens.

The action generation process during evaluation can be formalized as:

$$\hat{a}_t^i \sim P_\theta(a_t^i | s_{\leq t}, a_{<t}^{<i}, r_t) \tag{5}$$

In Equation 5, a_t^i represents the i-th action at timestep t. Actions are generated autoregressively, allowing the model to produce any number of actions for the latest timestep. Importantly, the action embedding and action head shapes remain consistent across downstream tasks.

Following the **ActionTokenizedSpreadEmbedding** strategy, a single timestep of actions with dimension d_a is expanded to include an additional $d_a - 1$ values. This action dimension expansion is distinct from flattening, as it remains ambiguous whether actions from the same timestep or index are interleaved (e.g., $a_0^0, a_0^1, \dots$ versus $a_0^0, a_1^0, \dots$).

After this spreading and interleaving, the resulting sequence conforms to Equation 6, as opposed to Equation 1. The training regimen for this method is detailed in Algorithm 1. A comparison of the performance of both approaches as compared to the original DT model (which we refer to as the baseline) is presented in the results section.

$$\boldsymbol{r_n, s_n, a_n^i} = r_0, s_0, a_0^0, a_0^1, \dots, r_1, s_1, a_1^0, \dots, \tag{6}$$

Where here a_n^i is the i-th feature of the action at timestep n.

Downstream Tasks

One of the aims of this research is to facilitate adaptation to new modalities or environments for downstream tasks with minimal alterations to the model architecture while preserving the state space (meaning no tokenization of the state and reward inputs similar to TT). While the observation spaces may differ across environments, the action embedding layer can remain consistent, requiring only a new state embedding layer (and action head in the case of **ActionTokenizedEmbedding**).

We opt to train new embedding layers explicitly but also recognize the potential in methods similar to (Reid, Yamada, and Gu 2022), which could align new layers with pretrained ones, or use unsupervised learning techniques (Girdhar et al. 2023). When actions are not expanded in the tuple, a new action head must be trained for the modified observation and action space, but the pre-existing embedding layer can be preserved.

This approach aligns with the emerging trend toward foundational models in RL, enabling more efficient training on novel tasks with limited data. This is analogous to the recent development of large language models (LLMs) (Touvron et al. 2023), which have offered a cost-effective starting point for research, previously attainable only through significant computational investment.

Algorithm 1: Action Tokenized Spread DT

Input: H model and H_r, H_s, H_a, H_t modality embedding layers, T tokenization schema, n_i training iterations

Output: H_θ model

Data(*Dataset* $\mathcal{D}_{\text{offline}}$, *trajectories* τ $(s_t^{d_s}, a_t^{d_a}, r_t)$)

for $n = 1, ..., n_i$ **do**

  /* draw batch from the dataset */

  $\tau \leftarrow \mathcal{D}_{\text{offline}}$

  /* tokenize the actions */

  $a' \leftarrow T(a_t^{d_a})$

  /* embed input modalities */

  $E_r, E_s, E_a \leftarrow H_r(r), H_s(s), H_a(a')$

  /* position & timestep embeddings */

  $E_t \leftarrow H_t(t)$

  $E_r, E_s, E_a \leftarrow E_r + E_t, E_s + E_t, E_a + E_t$

  /* actions into modality dimension */

  $E'_a \in \mathbb{R}^{(T \cdot d_a) \times e} \leftarrow E_a \in \mathbb{R}^{T \times d_a \times e}$

  /* interleave inputs along T */

  $E \leftarrow \text{concat}(E_r, E_s, E'_a)$

  /* forward and backward pass */

  $H_\theta \leftarrow \text{update}(H, \mathcal{L}_{mse}(a, H(E)))$

Experimental Setup

We follow similar setups and experimental runs to the benchmarks established in previous works (Chen et al. 2021; Zheng, Zhang, and Grover 2022; Janner, Li, and Levine 2021) and the Clean Offline RL (CORL) codebase (Tarasov et al. 2023). All of these works primarily evaluate on mujoco environments (e.g. Antmaze, HalfCheetah, Hopper, Walker2d) that come with the D4RL offline dataset (Fu et al. 2021) that have various number and length trajectories as de-

scribed in the D4RL paper (Fu et al. 2021). We run the experiment multiple times (4 times each with a 100 vectorized environments) with different seeds and average the results across runs.

As our research is focused on the DT architecture and many of the algorithms compared against in previous works are not amenable to the desired downstream configurations we are interested in, we compare our results to the baseline DT model which we label as ActionEmbedding throughout the results. As the DT uses a conditioned reward in a way to control what sort of actions to generate (e.g. a lower reward will in theory result in the model predicting worse actions that generate lower expected rewards) we run our experiments with the available conditioned rewards when they are available from the CORL codebase (meaning they do not come from the D4RL dataset, rather they generally are a value close to the maximum reward seen during training and a value that is approximately half of the maximum reward seen during training). To run experiments with a model for a downstream tasks we require a model that has done pretraining on any prior task. For this we select a single individual task that is not the same environment or dataset as the downstream task and train the model on that task for 50k iterations. The number of downstream-to-pretrain environments (and not even considering the dataset) is exponentially large so we choose to only run a subset of these experiments. In future work we aim to utilize generalized pretraining regimes (Sun et al. 2023) to further reduce the number of experiments that need to be run but note that these experiments require significant more computational resources (due to the model size growing significantly to accommodate the number of environments, similar to what was shown in (Reed et al. 2022)).

Results

Our experiments examine and shine light on the following:

(i) Can tokenization be applied to only actions for a trajectory so that fewer layers of the model must be adapted for new environments/modalities?

(ii) To what extent does the granularity of tokenization or tokenization across features impede model performance?

(iii) Can the model adapt to new modalities with little or no supervised training and still perform better than random?

(i) Training Comparison

The first question is examined by comparing the performance of the baseline DT to the tokenized actions with both the ActionTokenizedEmbedding and ActionTokenizedSpreadEmbedding methods. Along with our proposed methods, we also include a model that only spreads the action vector into the modality dimension but does not tokenize the actions (ActionSpreadEmbedding) which achieves good performance but does not allow embedding layer reuse for downstream environments. As seen in Figure 1, the model with the tokenized actions performs as well as the baseline DT model. We show a comparison for all of the standard mujoco environments in the appendix and find only AntMaze to be noticeably slower to train but still achieves the same performance.

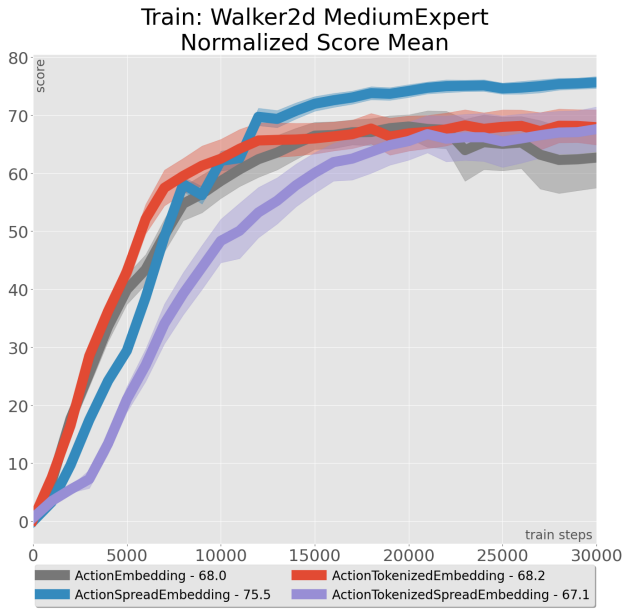

Figure 1: Training comparison of baseline DT (ActionEmbedding) versus tokenized actions for the Walker2d environment. The value following the name in the legend is the max value for the averaged runs.

We include additional model configurations for this evaluation including showing the impact of no tokenization offset (meaning that the token range for a_i and a_{i+1} are the same) as well as the use of an action index embedding rather than the action index offsetting (as this would greatly decrease the embedding size for the action embedding layer since the offset comes from the action index embedding). What we see in Figure 1 as well as for the other mujoco environments is that using a tokenization schema which gives a unique range of tokens per action feature results in the model performing as well as the baseline DT.

We also tested incorporating an additional loss metric that embeds the states into the output sequence but found no benefit by incorporating additional loss metrics in any capacity.

Since many of the features in the action space share the same underlying unit of measurement (e.g. torque or force) it seems plausible that they could share the same tokenization value per feature but we find this is not the case. This is important for many reasons, one of which is that model size can increase quickly due to the embedding layer when compared to the baseline and computational limits are a consideration of our research.

Many of the datasets used for offline learning contain a bimodal distribution, as seen in Figure 2. We also note that due to the distribution of actions, as seen in Figure 2, the tokenization or discretization schema is quite important and we utilize a quantile tokenization schema as opposed to uniform binning (the offline datasets all have actions between -1 and 1).

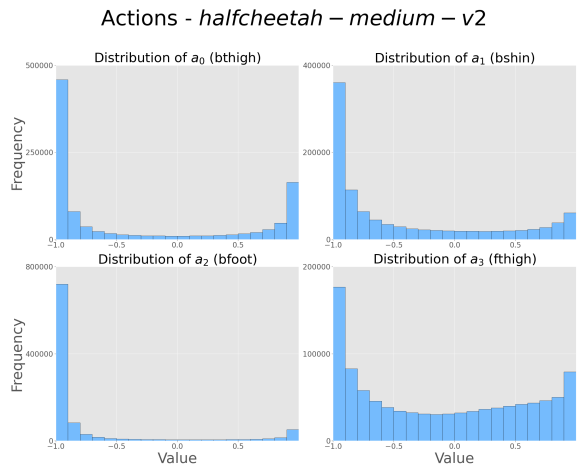

Figure 2: Histogram plot of the Actions for the Halfcheetah Environment Medium-v2 Dataset showing how the distribution of these actions is bimodal with modes generally at ± 1.

(ii) Quantization Fidelity

In terms of quantization fidelity (i.e. how many tokens should the action values be binned into), we compare model performance while varying the number of bins as seen in Figure 3. We vary the actions from 100 to 3000 bins while tokenizing with the included per action offset from Equation 4. As the actions are tokenized using a quantile method (i.e. the bins are not uniform), the increase of bins results in a increase of fidelity around the modes of the bimodal distribution (generally ± 1) of the actions and presumably increasing the fidelity around these bimodal peaks adds little extra value (we also found quantile tokenization always outperforms uniform tokenization which we omit).

While there is noticeable difference in performance when comparing the number of bins for the ActionTokenizedSpreadEmbedding model, we also note that for the ActionTokenizedEmbedding model there is little to no difference in performance when comparing the number of bins as seen in Figure 4. Here we similarly plot an average of 5 runs (where each run is 100 vectorized environments) while varying the number of bins from 100 to 3000. We see that the performance of the model is roughly the same for all of the number of bins and that the model is able to achieve a normalized score of between 101.7 (for the case of 3000 bins) and 105.3 (for the case with 1000 bins as well as the baseline ActionEmbedding model).

Our results indicate that the quantization to more bins is not de-facto better but due to the increase in number of bins resulting in larger embedding layers (which increases the model size), the time taken for performance to saturate (i.e. the number of training iterations) increases on these models that incorporate greater specificity from tokenization. As limited compute is generally the case when training only the embedding related layers of a foundation model, we suggest keeping the number of bins to a minimum.

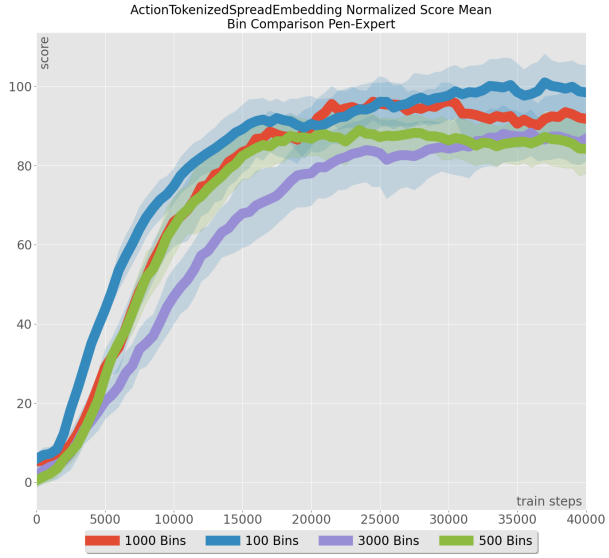

Figure 3: Comparison of number of bins for Pen environment. Here we see a general trend that more bins increases number of training iterations needed to achieve roughly the same performance. Although we left out the baseline model and the bin comparison of the Action Tokenized Embedding model here, both achieve a normalized score with a mean of 109.5 and 110.1 respectively.

(iii) Downstream Tasks

The goal with this research is aimed at how to generalize the DT architecture which can then be used with minimal fine-tuning on a new environment, we setup experiments that evaluate the scenario where a model is initially trained using a dataset associated with one environment, and then fine-tuned on a downstream task of a novel environment (i.e. not just a novel task within that environment).

As the state and action space of the downstream environments is different from that seen during pretraining, we compare the various methods outlined above while keeping the training regimen the same from model to model (i.e. only allowing specific modalities of the embedding layers to be trained or allowing all layers to be trained).

With this we see as in Figure 5 that when we are using a pretrained model where only the embedding layers are trained on a new downstream dataset, tokenization results in a model with a higher normalized score than the baseline model (ActionEmbedding). In this plot, while both the ActionTokenizedEmbedding and ActionTokenizedSpreadEmbedding model outperform the baseline model, the ActionTokenizedSpreadEmbedding model is slower to train but results in lower variance in the normalized score. This suggests that the tokenization and spreading of the actions into the modality dimension allows the model to translate the new environment actions into a format that is more similar to the pretraining environment actions albeit at a slower pace than the ActionTokenizedEmbedding model.

In our comparative analysis for downstream tasks, we evaluate the performance impact when all layers are allowed

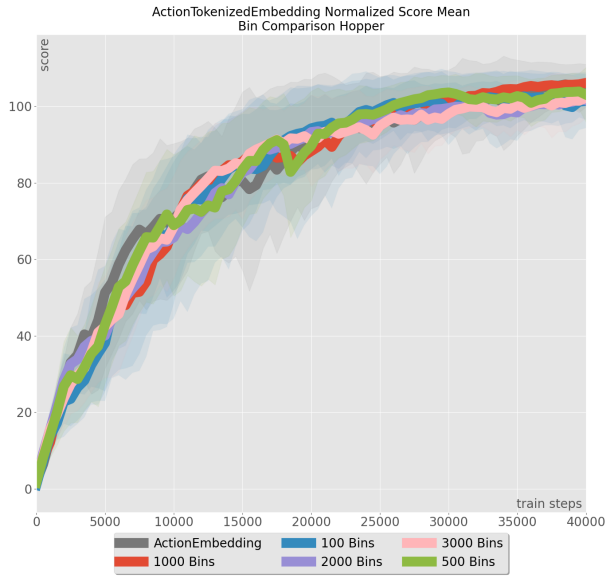

Figure 4: Comparison of number of bins for Hopper environment. In contrast to Figure 3 there is little impact on performance when using an Action Tokenized Embedding model when compared to the baseline Action Embedding model.

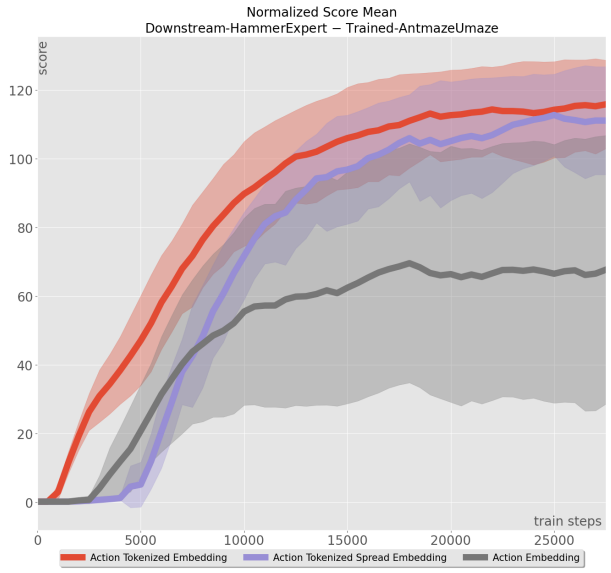

Figure 5: Normalized scores for downstream task hammer with expert dataset allowing updates to only embedding layer during downstream training. Pretraining done with the Antmaze umaze dataset.

to be trained (versus fine-tuning only the modality specific embedding layers). Shown in Figure 6 we see the results for the Walker2d environment when all layers are allowed to be updated and we observe that the Action Tokenized Spread Embedding model outperforms the Action Embedding and Action Tokenized Embedding model by 11 and 13 points respectively for the normalized score. This outcome not only underscores the potential of the Action Tokenized Spread Embedding model to enhance reinforcement learning tasks but also suggests the value of our tokenization strategy in facilitating more efficient learning and adaptation for future foundational RL models.

In contrast to allowing all layers to be trained for the downstream task, we also show the case where only the embedding layers are trained on the downstream task which is a likely setup for using large general pretrained models that due to memory constraints may not be trained where all layers can be updated (and similar in setup to what was shown in Figure 5).

This setup is shown in Figure 7 where the Action Embedding has new embedding layers specific to the downstream environment performs substantially worse than the ActionTokenizedSpreadEmbedding model. Here, the Action Embedding model (which is equivalent to the DT) is barely able to learn on this downstream task while the model with the tokenized spread actions is able to learn and achieves a score that although it is 72% below the fully trained model, is 21 normalized points above its baseline comparison.

Unfortunately, for the offline datasets that may achieve poor results during evaluation in the environment (e.g. the door environment has three offline datasets, human-v1, cloned-v1, and expert-v1, where the human-v1 and cloned-

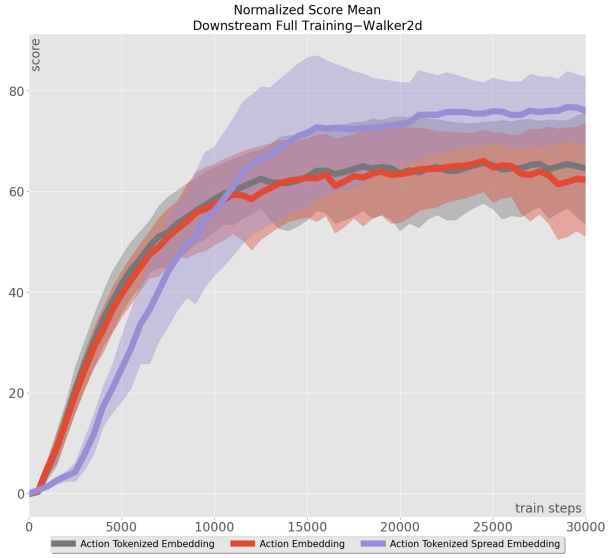

Figure 6: Downstream training on Walker2d (pretrained with Hopper medium expert dataset) allowing all layers to be trained. In this experiment the Action Tokenized Spread Embedding model outperforms the Action Embedding and Action Tokenized Embedding model by 11 and 13 points respectively for the normalized score.

v1 both achieve normalized scores less than 10 while the model trained with the expert-v1 dataset achieves a normalized score of 100 which is similar to others results (Tarasov et al. 2023)), the tokenized spread embedding model is unable to learn at all as seen in Figure 8.

In this experiment, the Action Tokenized Embedding is

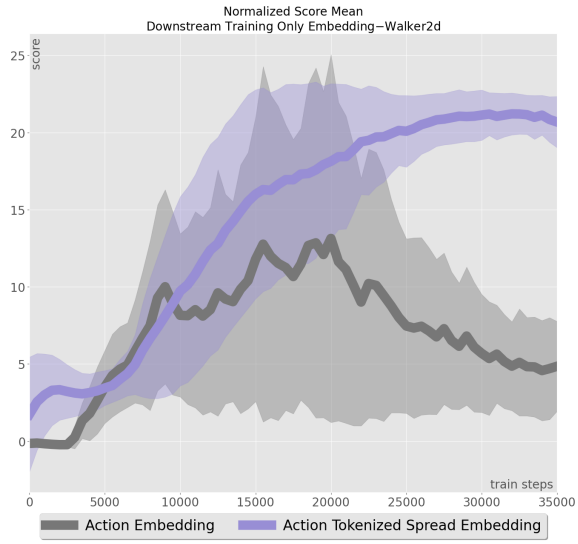

Figure 7: Comparison between the embedding layers only downstream training on Walker2d (pretrained with hopper medium expert dataset for the Action Embedding and Action Tokenized Spread Embedding models. When unable to train the entire model (which is likely to happen due to memory constraints when training larger foundational models) the Action Embedding model performs significantly worse than the Action Tokenized Spread Embedding model.)

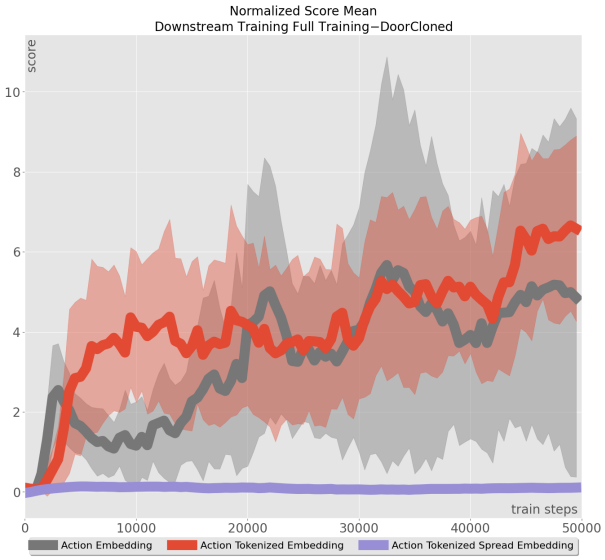

Figure 8: Downstream training on door for model trained with Door/Adroit environment with the cloned dataset. Here we see that while the Action Embedding and Action Tokenized Embedding model both have quite a low normalized score, the Action Tokenized Spread Embedding model performance is near 0.

still able to generally outperform the baseline Action Embedding model with its main difference being that the eval has lower variance, for example in Figure 8 the Action Embedding $\sigma = 4.4$ while for the Action Tokenized Embedding $\sigma = 2.1$. The observed decrease in variance is possibly due to the tokenization offering a sort of noise filter on the actions, which is particularly useful in environments with some stochasticity (which these offline datasets have). Applying tokenization in some manner appears particularly advantageous in environments characterized by noisy actions, as it aids the model in integrating the noisy data into sequences that the model can use in a more consistent manner.

We do note that this effect is lost for poor performing environments as seen in Figure 8 where the normalized scores are quite low implying this effect is best noticed on models that are already performing well and will not improve the baseline results.

Conclusion

In this work we analyzed methods to enhance the adaptability and efficiency of Reinforcement Learning (RL) models by incorporating tokenization schemes tailored to specific types of input. Our findings suggest that the strategy of tokenization and "spreading" actions can often facilitate more efficient training for subsequent tasks but may have drawbacks such as slower training and low performance in environments when the model would otherwise not be performative. Our methodology provides a likely more generalized foundation for RL models, drawing parallels to the versatility observed in language models, as noted in recent research (Reed et al. 2022; Touvron et al. 2023) whereby models are easily able to adept to downstream tasks. Moreover, this work echoes the broader vision of foundational models in RL, whereby one core training schema can be adapted across tasks, requiring minimal fine-tuning. This paves the way for RL models that are as adaptable and efficient as contemporary large language models.

Our aspiration is to motivate researchers with the computational capacity to develop and distribute models that possess general RL capabilities (rather than for specific environments which is generally the case), thereby supporting the broader research community, particularly those limited by computational constraints, in the pursuit of large generalized RL models. We anticipate that these expansive models will exhibit emergent properties akin to those in language models, such as zero-shot learning and the proficiency to acclimate to new challenges with minimal fine-tuning.

References

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. arxiv:2106.01345.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2021. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. arxiv:2004.07219.

Gao, C.; Wu, C.; Cao, M.; Kong, R.; Zhang, Z.; and Yu, Y. 2023. ACT: Empowering Decision Transformer with Dynamic Programming via Advantage Conditioning. arxiv:2309.05915.

Girdhar, R.; El-Nouby, A.; Liu, Z.; Singh, M.; Alwala, K. V.; Joulin, A.; and Misra, I. 2023. ImageBind: One Embedding Space To Bind Them All. arxiv:2305.05665.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arxiv:2106.09685.

Janner, M.; Li, Q.; and Levine, S. 2021. Offline Reinforcement Learning as One Big Sequence Modeling Problem. In *Advances in Neural Information Processing Systems*.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; de las Casas, D.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; Lavaud, L. R.; Lachaux, M.-A.; Stock, P.; Scao, T. L.; Lavril, T.; Wang, T.; Lacroix, T.; and Sayed, W. E. 2023. Mistral 7B. arxiv:2310.06825.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. arxiv:2005.01643.

Meng, L.; Wen, M.; Yang, Y.; Le, C.; Li, X.; Zhang, W.; Wen, Y.; Zhang, H.; Wang, J.; and Xu, B. 2022. Offline Pre-trained Multi-Agent Decision Transformer: One Big Sequence Model Tackles All SMAC Tasks. arxiv:2112.02845.

Paster, K.; McIlraith, S.; and Ba, J. 2022. You Can't Count on Luck: Why Decision Transformers and RvS Fail in Stochastic Environments. arxiv:2205.15967.

Reed, S.; Zolna, K.; Parisotto, E.; Colmenarejo, S. G.; Novikov, A.; Barth-Maron, G.; Gimenez, M.; Sulsky, Y.; Kay, J.; Springenberg, J. T.; Eccles, T.; Bruce, J.; Razavi, A.; Edwards, A.; Heess, N.; Chen, Y.; Hadsell, R.; Vinyals, O.; Bordbar, M.; and deFreitas, N. 2022. A Generalist Agent. arxiv:2205.06175.

Reid, M.; Yamada, Y.; and Gu, S. S. 2022. Can Wikipedia Help Offline Reinforcement Learning? arxiv:2201.12122.

Shi, R.; Liu, Y.; Ze, Y.; Du, S. S.; and Xu, H. 2023. Unleashing the Power of Pre-trained Language Models for Offline Reinforcement Learning. arxiv:2310.20587.

Sun, Y.; Ma, S.; Madaan, R.; Bonatti, R.; Huang, F.; and Kapoor, A. 2023. SMART: Self-supervised Multi-task pretrAining with contRol Transformers. arxiv:2301.09816.

Tarasov, D.; Nikulin, A.; Akimov, D.; Kurenkov, V.; and Kolesnikov, S. 2023. CORL: Research-oriented Deep Offline Reinforcement Learning Library. arxiv:2210.07105.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arxiv:2307.09288.

Wang, K.; Zhao, H.; Luo, X.; Ren, K.; Zhang, W.; and Li, D. 2022. Bootstrapped Transformer for Offline Reinforcement Learning. arxiv:2206.08569.

Yamagata, T.; Khalil, A.; and Santos-Rodriguez, R. 2022. Q-Learning Decision Transformer: Leveraging Dynamic Programming for Conditional Sequence Modelling in Offline RL. arxiv:2209.03993.

Zhao, Y.; Gu, A.; Varma, R.; Luo, L.; Huang, C.-C.; Xu, M.; Wright, L.; Shojanazeri, H.; Ott, M.; Shleifer, S.; Desmaison, A.; Balioglu, C.; Damania, P.; Nguyen, B.; Chauhan, G.; Hao, Y.; Mathews, A.; and Li, S. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arxiv:2304.11277.

Zheng, Q.; Zhang, A.; and Grover, A. 2022. Online Decision Transformer. arxiv:2202.05607.