ACCELERATING TRANSFORMERS IN ONLINE RL

Anonymous authors
Paper under double-blind review

ABSTRACT

The appearance of transformer-based models in Reinforcement Learning (RL) has expanded the horizons of possibilities in robotics tasks, but it has simultaneously brought a wide range of challenges during their implementation, especially in model-free online RL. Most existing learning algorithms cannot be easily implemented with transformer-based models due to the instability of the latter. In this paper, we propose a method that uses the Accelerator agent as a transformer's trainer. The Accelerator trains in the environment by itself and simultaneously trains the transformer through behavior cloning during the first stage of the proposed algorithm. In the second stage, the pretrained transformer starts to interact with the environment in a fully online setting. As a result, this algorithm accelerates the transformer in terms of its performance and helps it to train online more stably.

1 INTRODUCTION

Transformers (Vaswani et al., 2017) have demonstrated remarkable success in various domains due to their ability to model long-range dependencies and complex patterns. Another benefit of transformers is their suitable architecture for processing multi-modal data, such as text and images (Kim et al., 2024; Radford et al., 2021). All these properties contribute to the creation of multi-task agents (Jiang et al., 2022; Reed et al., 2022; Team et al., 2024). There are three main approaches to training transformers in reinforcement learning: 1) fully offline training (Chen et al., 2021), 2) offline pretraining with further online fine-tuning (Sun et al., 2023), and 3) fully online training without offline data (Parisotto et al., 2020; Pramanik et al., 2023).

Offline Reinforcement Learning (RL) (Chen et al., 2021; Janner et al., 2021) is a widely used approach for training transformer models. It is beneficial to use offline data to lower training costs, which is particularly important in fields such as robotics or autonomous transportation. Despite its advantages, offline RL has several limitations that hinder agent performance and create barriers to its widespread adoption. One of the main weaknesses is the effort required to collect expert demonstrations. Sophisticated environments often require large models, which, in turn, demand a significant amount of training data to perform well. Moreover, to maintain high performance during evaluation or real-world deployment, attention must be paid to distributional shifts between offline trajectories and the real state distribution. Another drawback of offline RL is its inability to regulate the exploration process, which can potentially limit agent performance and underscores the need for a large and high-quality dataset.

However, their application in online RL remains challenging too, primarily due to instability issues during training (Parisotto et al., 2020). Transformers are known to be sensitive to hyperparameters and optimization settings, which makes their training process inherently unstable (Parisotto et al., 2020). In online RL, where the agent interacts with the environment in real-time, this instability is exacerbated by the non-stationary nature of the data distribution. Transformers typically require large amounts of data to achieve good performance. In online RL, where data is collected incrementally through interactions with the environment, this poses a challenge. The agent must learn efficiently from limited and potentially noisy data, which contrasts with the data-hungry nature of transformers.

Our proposed approach addresses problems such as training instability, sample inefficiency, and exploration limitations, and avoids the need for offline data. Despite the fact that the first stage of our method utilizes BC from the Accelerator's demonstrations, it still maintains the ability to explore the environment. Our method avoids environment distribution shifts by letting the transformer explore the environment by itself in the second stage of the training procedure. The design of the accelerator depends on the nature of the task: whether the environment can be described by a Markov Decision Process (MDP) or a Partially Observable Markov Decision Process (POMDP), or whether we process a vector-based environment or an image-based environment.

In this paper, we propose an algorithm that pretrains (accelerates) a transformer in an online manner, conduct experiments on MDP vector-based robotic locomotion (Todorov et al., 2012) and manipulation (Tao et al., 2024) tasks, and compare the accelerated transformer with MLP and LSTM baselines.

Our contribution is as follows:

1. We propose a flexible and easy-to-tune transformer training approach that effectively pretrains a transformer in an online manner, eliminating the need for an offline dataset.
2. By providing thorough experiments, we empirically demonstrate the effectiveness of the proposed algorithm on robotic locomotion and manipulation tasks.

2 RELATED WORK

There is a growing body of work that utilizes transformers in RL, adapting their architecture and training algorithms to ensure stable performance. Chen et al. (2021) proposed the Decision Transformer (DT), which reduces the RL task to supervised sequence modeling. Janner et al. (2021) enhance simple sequence modeling by utilizing beam-search, a technique originally from Natural Language Processing (NLP), in order to improve the generation of action trajectories. However, offline data can consist of sub-optimal trajectories, so sequence modeling approaches could affect policy performance. To address this problem, Q-learning Decision Transformer (Yamagata et al., 2023) leverages dynamical programming (especially Q-learning) and relabel return-to-go in order to train DT on new data. Another problem of the offline RL is a limited quantity of the training data. To overcome this problem, Wang et al. (2022) proposed the Bootstrapped Transformer which relies on bootstrapping ideas and generate synthetic dataset for trained model.

Offline pretraining with further online fine-tuning eliminates some of the problems of the previous approach, especially since it can regulate exploration during online interaction with the environment. SMART algorithm (Sun et al., 2023) separates the training process into two steps: 1) Offline self-supervised sequence modeling and 2) Online fine-tuning. Nair et al. (2020) proposed AWAC, an algorithm that enables rapid fine-tuning with a combination of prior demonstration data and online experience. Chan et al. (2024) proposed a novel training algorithm that can learn from a small amount of demonstrations, while classical Behavior Cloning (BC) requires more data to achieve the same result. The authors of the Online Decision Transformer (Zheng et al., 2022) use an offline dataset and minimize the log-likelihood of expert demonstrations in order to stabilize the agent during online training. This technique uses pre-collected data simultaneously with online training.

Fully online training turned out to be a more complex and problematic task in transformer-based RL. Based on the Transformers-XL (TrXL) architecture (Dai, 2019), Gated Transformer-XL (Parisotto et al., 2020) (GTrXL) uses gating, learnable functions that can regulate the proportion of bypass information from skip-connections. A combination of the TrXL properties and gating capabilities makes GTrXL able to train well in memory-demanding tasks Pleines et al. (2023).

3 PROPOSED METHOD

The effects associated with pretraining transformers on offline datasets can have negative consequences, such as limited exploration of the environment and state distribution shift. Our proposed method avoids these drawbacks by pretraining the transformer in an online manner via behavior cloning and additional gradient ascent over the critic's function.

Our method consists of two stages. The key idea is to utilize the accelerator's stability to provide a stable learning process for the transformer. Although this section provides an explanation based on an **MLP accelerator**, any other model design choices are still compatible and can be used instead. The main rule is the following: the accelerator may have weaker performance on the environment,

Algorithm 1 TD3-based **acceleration** stage algorithm

```
1: Initialize accelerator's critics $Q_{\phi_1}, Q_{\phi_2}$, actor $\pi_\phi$, and transformer actor $\pi_\theta$
2: Initialize target networks $\phi'_1 \leftarrow \phi_1, \phi'_2 \leftarrow \phi_2, \phi' \leftarrow \phi, \theta' \leftarrow \theta$
3: Initialize replay buffer $\mathcal{B}$ and trajectory buffer $\mathcal{T}$
4: for $t = 1$ to $T$ do
5:     Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
6:     $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
7:     Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$, Store tuple $(s, a)$ in $\mathcal{T}$
8:     Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
9:     $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
10:    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\phi'_i}(s', \tilde{a})$
11:    Update critics $\phi_i \leftarrow \arg\min_{\phi_i} N^{-1} \sum (y - Q_{\phi_i}(s, a))^2$
12:    Train transformer by applying algorithms 2 and 3
13:    $\mathcal{T} = \emptyset$
14:    if $t$ mod $d$ then
15:        Update $\phi$ by the deterministic policy gradient:
16:        $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\phi_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
17:        Update target networks:
18:        $\phi'_i \leftarrow \tau\phi_i + (1 - \tau)\phi'_i$
19:        $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
20:    end if
21: end for
```

but it must be more stable in terms of training. This ensures that the accelerator will bring the transformer to a level sufficient for further online fine-tuning. Though the explanation is based on the Twin Delayed Deep Deterministic Policy Gradient (Fujimoto et al., 2018) (**TD3**) algorithm, it can be generalized to other off-policy actor-critic algorithms such as Deep Deterministic Policy Gradient (Lillicrap, 2015) or Soft Actor-Critic (Haarnoja et al., 2018).

3.1 THE FIRST STAGE: TRANSFORMER ACCELERATION

In the first stage, the accelerator policy trains by itself while simultaneously serving as a trajectory generator for the transformer policy. Let us define the accelerator agent as $\pi_\phi(a|s)$ and the transformer agent as $\pi_\theta(a|s)$. During the entire first stage, $\pi_\phi(a|s)$ trains via the standard RL pipeline, with the exception that it saves states and actions in a special trajectory buffer $\mathcal{T}$, which is then used as a dataset to train the transformer.

Algorithm 1 describes the process of transformer acceleration based on the TD3 training pipeline. All changes made to the standard TD3 algorithm are highlighted in red. In particular, rows 5, 6, 7, and 12 clarify how trajectories are collected in $\mathcal{T}$ and used for transformer acceleration. Note that this approach can also be applied during the accelerator's evaluation process.

During the transformer's update phase, it can utilize either behavior cloning alone or an additional gradient ascent method, as described in Algorithm 2 and Algorithm 3, respectively. Since the accelerator is trained by an actor-critic algorithm, we want the actor π_ϕ to ascend its critic's function, which approximates the state-action function $Q_{\phi_i}(s, a)$. Therefore, it becomes possible to improve transformer training by adjusting its weights in the direction of the critic's gradient ascent, as in Algorithm 3.

During the first stage, the transformer agent forms its basic structure of weights that will assist it in online training. This approach leverages supervised learning, which is beneficial in terms of the transformer's stability.

Notably, since we generate our transformer's training data in real time, we can regulate its quality and diversity by dynamically changing the parameters of the Accelerator's training. For instance, to increase exploration, we can add Gaussian noise to the states from the environment, $\tilde{s} = s + \mathcal{N}(0, \sigma)$, sample target actions according to these observations, $a \sim \pi_\phi(\tilde{s})$, store $(\tilde{s}, a)$ in $\mathcal{T}$, and use this data within Algorithm 2. In addition, the length of the training session can be as long as needed, so the volume of the training data could theoretically be unlimited.

These measures help overcome well-known offline RL challenges such as the exploration problem, dataset size limitations, and state distribution shifts.

Algorithm 2 Behavior cloning

```
Require: Transformer $\pi_\theta(a|s)$, $\mathcal{T}$
for $s, a$ in $\mathcal{T}$ do
    Make prediction $\hat{a} = \pi_\theta(s)$
    Calculate $L = MSE(a, \hat{a})$
    Update actor $\theta_{t+1} = \theta_t - \alpha \nabla_\theta L$
    Update target $\theta'_{t+1} \leftarrow \tau \theta_{t+1} + (1 - \tau)\theta'_t$
end for
```

Algorithm 3 Ascending on critic

```
Require: Transformer $\pi_\theta(a|s)$, $\mathcal{T}$, critic $Q_{\phi_1}$
for $s, a$ in $\mathcal{T}$ do
    Make prediction $\hat{a} = \pi_\theta(s)$
    Calculate $Q_{\phi_1}(s, \hat{a})$ via accelerator's critic
    Update $\theta_{t+1} = \theta_t + \alpha \nabla_{\hat{a}} Q(s, \hat{a}) \nabla_\theta \hat{a}$
    Update target $\theta'_{t+1} \leftarrow \tau \theta_{t+1} + (1 - \tau)\theta'_t$
end for
```

Architecture and training details are available in Table 1.

3.2 THE SECOND STAGE: ONLINE FINE-TUNING

The first stage yields pretrained transformer policy which is ready to continue its training in the fully online setting. At this stage, we can drop accelerator's actor but continue using its critic in order to conjugate it with the transformer actor. Starting from this time, these two models operate together in casual online RL training pipeline, described in algorithm 4.

Algorithm 4 TD3-based **fine-tuning** stage algorithm

```
Use $Q_{\phi_1}$, $Q_{\phi_2}$,$\pi_\theta$ with their targets from the first stage, initialize empty replay buffer $\mathcal{B}$
for $t = 1$ to $T$ do
    Select action with exploration noise $a \sim \pi_\theta(s) + \epsilon$,
    $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$
    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\theta'}(s') + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\phi'_i}(s', \tilde{a})$
    Update critics $\phi_i \leftarrow \arg\min_{\phi_i} N^{-1} \sum (y - Q_{\phi_i}(s, a))^2$
    if $t$ mod $d$ then
        Update $\theta$ by the deterministic policy gradient:
        $\nabla_\theta J(\theta) = N^{-1} \sum \nabla_a Q_{\phi_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)$
        Update target networks:
        $\phi'_i \leftarrow \tau \phi_i + (1 - \tau)\phi'_i$
        $\theta' \leftarrow \tau \theta + (1 - \tau)\theta'$
    end if
end for
```

Algorithm 4 describes the fine-tuning stage of the transformer. As mentioned earlier, the acceleration stage returns three neural networks: the accelerator actor $\pi_\phi(a|s)$, the accelerator critic Q_{ϕ_i}, , $(i = 1, 2)$, and the transformer $\pi_\theta(a|s)$. During the fine-tuning stage, we continue to use the accelerator critic alongside the transformer for further online training. In our TD3-based algorithm description, the fine-tuning stage represents a standard training process using the same TD3 algorithm, where the actor is the transformer and the critic is the accelerator critic.

As we previously noted, the advantage of our algorithm lies in its complete flexibility when choosing architectures for the accelerator. This means that in a POMDP setting, it is possible to select a critic architecture capable of processing sequences of states, thereby preventing the critic's architectural limitations from affecting the transformer's fine-tuning stage.

4 EXPERIMENTS

4.1 ENVIRONMENTS AND BASELINES

We evaluate the proposed method on two types of vector-based environments: MuJoCo (Todorov et al., 2012) (HalfCheetah, Ant, and Hopper) and ManiSkill (Tao et al., 2024) (PushCube, PullCube). MuJoCo is a physics engine designed for research and development in robotics. It is used to simulate locomotion tasks, where agents learn to move efficiently in complex environments. ManiSkill is a simulation environment designed for robotic manipulation tasks, focusing on dexterous manipulation, object interaction, and task-oriented learning.

In this paper, we use MLP, LSTM (Hochreiter & Schmidhuber, 1997), and Vanilla Transformer (Vaswani et al., 2017) baselines trained from scratch using TD3, and compare the accelerated transformer's performance with them. We use the MLP because of its ease of learning and stability; it can achieve high episodic rewards on both the MuJoCo and ManiSkill tasks we use. Similar to the transformer, the LSTM architecture processes a sequence as input, but it is often easier to train in RL settings and is more robust to noisy or irregularly sampled data. These considerations motivate comparing the transformer with LSTM to see whether our algorithm allows us to train the transformer to achieve results comparable to those of the LSTM. Finally, comparing the accelerated transformer with an online-trained Vanilla Transformer helps illustrate the benefits of our approach.

All the charts in this section describe the training progress of the models in terms of evaluation reward (for MuJoCo environments) or evaluation success rate (for ManiSkill environments). Each progress curve also includes a standard deviation, obtained by averaging the key metric over 30 seeds with 1 parallel environment for MuJoCo tasks, and 1 seed with 50 parallel environments for ManiSkill tasks.

All experiments are conducted on a Tesla V100 GPU with 128GB of RAM. Additional information about the accelerators, transformers, and baseline models' parameters, as well as their corresponding training parameters, is available in Table 1.

4.2 RESEARCH QUESTIONS

This section is dedicated to addressing the research questions (RQ) that arose during the implementation of the proposed algorithm. Answers to these questions will help clarify the feasibility of applying this algorithm, its advantages compared to training a transformer from scratch without the acceleration phase, and provide insights into the nuances of tuning the training process to achieve maximum effectiveness. Within this section, we have identified four main questions:

1. Can the accelerated transformer be more stable than a transformer architecture trained online from scratch?
2. Can the transformer achieve performance comparable to MLP and LSTM baselines?
3. Does the additional gradient ascent (Algorithm 3) on the critic's function, which can be utilized during the acceleration stage, improve the quality/speed of the transformer's training?
4. Is the acceleration stage sufficient for successful training of the transformer without the need for a fine-tuning stage?

These questions aim to provide a comprehensive understanding of the proposed algorithm, its effectiveness, and its practical implications.

RQ1. Can transformer train well on the first stage? All the Figures in this section include the evaluation reward and success rate, reflecting the training progress of the accelerated transformer during the online fine-tuning stage, as well as the baselines that started training from scratch in an online setting. According to Figure 1, the growth rate of the transformer's performance is comparable to the growth rate of the accelerator's performance. It is evident that the transformer is less stable than its accelerator, which aligns with the general understanding of the process, as the transformer clones the accelerator's behavior during its training session, where actions are distorted by Gaussian noise for exploration. Nevertheless, this does not prevent the transformer from learning to operate in the environment at a sufficiently high level. As shown in the PullCube environment (left graph), the

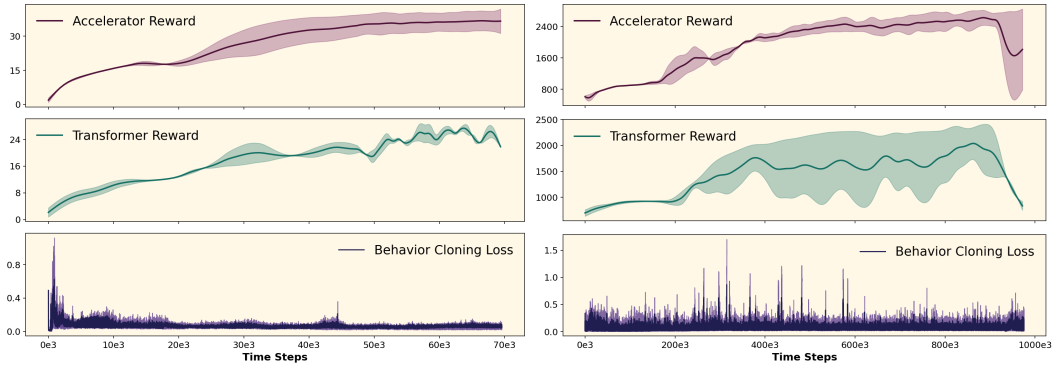

Figure 1: The first stage(transformers acceleration) in PullCube(left) and Ant(right) tasks. Top chart – accelerator's performance, middle chart – transformer's performance, bottom chart – behavior cloning loss.

peak success rate reaches 70%, which is a fairly good result. In the Ant environment (right graph), the transformer achieves an average reward of 2000, while its accelerator reaches around 2400.

Another important observation is how the transformer's performance closely mirrors the rises and falls in the accelerator's performance. On the PullCube graph, a slight dip in the accelerator's performance around the 20,000-step mark is accompanied by a similar behavior in the transformer's performance curve. Similar patterns are observed on the right graph (Ant environment), where adjustments to the accelerator at the end of training also lead to corresponding adjustments in the transformer's performance. It is clear that the first phase of acceleration is **insufficient** to obtain a fully trained transformer agent, which is why our algorithm utilizes a second phase for fine-tuning. Experiments in other environments can be found in Figure 5.

RQ2. Comparison with the baselines. In this section, we reveal the potential of the proposed algorithm and address whether the accelerated transformer can achieve performance comparable to MLP and LSTM. Figure 2 shows that the accelerated transformer reaches a performance competitive with MLP and LSTM in fewer training steps. According to the results in the HalfCheetah task (Figure 2, left), the transformer can be fine-tuned to surpass the performance of MLP and LSTM. Results in the PushCube task (Figure 2, right) demonstrate that it also requires significantly fewer steps to converge and achieve the maximum success rate.

In conclusion, the results of this and the previous section demonstrate the effectiveness of the proposed algorithm. Although the acceleration stage alone is insufficient to achieve peak performance, this can be compensated for by an online fine-tuning stage, during which the transformer exhibits stable, high-quality training. You can find supplementary materials for this experiment in Appendix A.

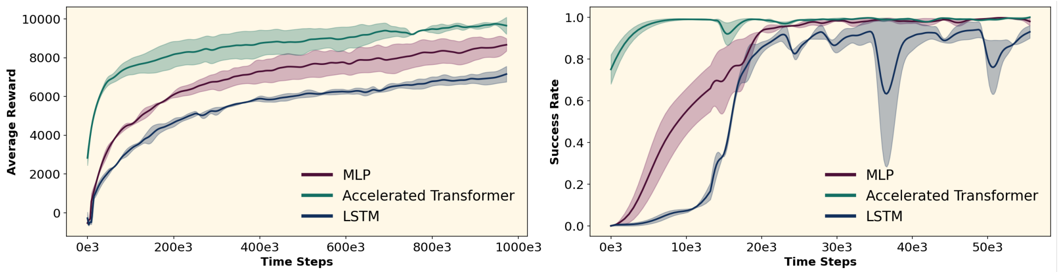

Figure 2: Average reward in HalfCheetah (left) and success rate in PushCube (right) tasks.

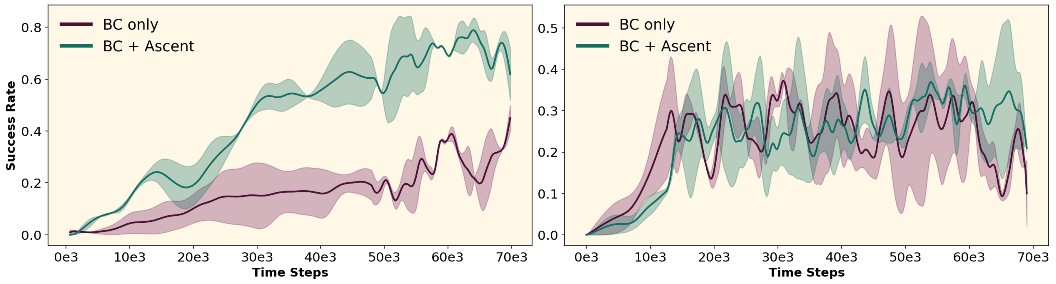

Figure 3: Success rate in PullCube (left) and PushCube (right) environments.

RQ3. Can additional gradient ascent improve acceleration? The Figures in this section present the transformer's evaluation reward or success rate during the acceleration stage, both with and without the additional ascent. According to subsection 3.1, there are two ways of training the transformer on trajectories from $\mathcal{T}$: by behavior cloning only or with additional gradient ascent over the critic's function. In this section, we compare these two options to determine whether additional ascent can improve acceleration. To ensure a fair comparison, we use fixed model and RL parameters listed in Table 1.

Figure 3 shows the transformer's acceleration progress during the first stage, both with and without additional gradient ascent. This technique provides a notable boost in the PullCube task, whereas in the PushCube environment it does not enhance the acceleration process.

Although training with additional ascent sometimes yields better performance, it does not guarantee improvement in every environment. To summarize, the use of additional gradient ascent does not ensure an improvement in the transformer's training; however, in some cases, it can prove beneficial and provide a performance boost. Each case should be evaluated individually to determine its effectiveness. You can find additional comparisons for this experiment in Figure 7.

RQ4. Why not just train the transformer from scratch? Figures in this section show the training progress of the MLP accelerator and the transformer during the first stage, allowing us to determine whether this stage alone is sufficient. A key question of this work is the feasibility of accelerating the transformer using the proposed algorithm, given that some of the MuJoCo and ManiSkill environments presented can be solved through online training of the transformer from scratch.

To address this question, it is necessary to consider the design of off-policy algorithms, which use a replay buffer for accumulating experience during the agent's training. To maintain high-quality training and the necessary level of exploration, it is essential to create large replay buffers, typically on the order of 1,000,000 observations (Fujimoto et al., 2018). Training the transformer on multiple environments in parallel, along with a large context, may require significant memory to store a large replay buffer. In image-based environments, the allocated memory increases even more due to the need to store images instead of vectors.

Thus, while online training from scratch is feasible for some environments, the proposed offline pretraining approach offers a potential solution to manage memory constraints and improve training efficiency, particularly in scenarios involving large-scale data and parallel environment interactions.

Our proposed algorithm addresses this issue, which can be particularly advantageous for low-power computers. Under the described transformer acceleration algorithm, the trajectory buffer $\mathcal{T}$ only needs to store fresh training data corresponding to the accelerator's current skills, eliminating the need to initialize $\mathcal{T}$ with a large size. As a result, throughout the entire acceleration stage, our memory costs for storing the trajectory buffer remain minimal.

Furthermore, during the online fine-tuning stage, the pre-trained transformer also does not require a large replay buffer. The need for extensive environment exploration has already been satisfied during the acceleration stage, so the replay buffer $\mathcal{B}$ does not need to store highly diverse experience. Consequently, the fine-tuning process can proceed efficiently with reduced memory overhead, making the algorithm well-suited for resource-constrained systems.

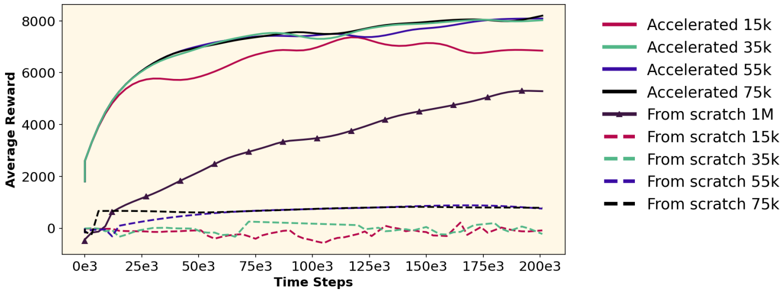

Figure 4: Comparison of the accelerated and non-accelerated transformer training performance with $\mathcal{B}$ of shape 15, 35, 55, 75 thousands and 1 million.

In the experiment shown in Figure 4, we ran both the accelerated and non-accelerated transformers with varying replay buffer sizes to highlight a key advantage of our algorithm. We selected replay buffer sizes of 15,000, 35,000, 55,000, and 75,000, and trained each model with these sizes. Additionally, to demonstrate the training quality of the non-accelerated transformer, we included a replay buffer size of 1,000,000. The results clearly show that the non-accelerated version failed to train effectively with any of the smaller buffer sizes, except for the 1,000,000-sized one. In contrast, our model demonstrated effective training with buffer sizes of 35,000, 55,000, and 75,000, and only began to slightly lose quality at the size of 15,000. These findings indicate that the effective threshold for the replay buffer size lies between 15,000 and 35,000, which is more than 40 times smaller than 1,000,000.

5 CONCLUSION

Training transformers in a fully online setting, as well as utilizing offline data for pre-training, faces challenges such as unstable training, limited datasets, weak exploration, and state distribution shifts. Our proposed method addresses these issues, enabling more stable and effective training of transformer-based models. It involves collecting data in an online manner while continuously training the expert policy, thereby mitigating the problems of limited data and state distribution shifts. The ability to dynamically adjust the expert policy's training parameters further enables environment exploration. The two-stage design of our algorithm allows fine-tuning the accelerated transformer in an online setting, which helps elevate the agent's performance to a higher level.

In this paper, we successfully tested the proposed algorithm on control tasks, achieving performance comparable to MLP and LSTM baselines. However, further development is needed to adapt it to image-based and POMDP environments. Additionally, we investigated the impact of additional gradient ascent on transformer acceleration and demonstrated that, in some cases, this technique can enhance the algorithm's quality. We also highlighted a significant advantage of our approach: the ability to reduce the replay buffer size by orders of magnitude, making the training of computationally intensive models like transformers more accessible for low-resource systems.

The goal of this work is to introduce the concept of transformer acceleration rather than limit it to the specific algorithm described here. This approach can take many forms, some of which may prove equally or even more effective. The proposed concept has potential for further development and for gradual refinement of the techniques used in acceleration. We hope that our work will inspire further advances in this direction.

REFERENCES

Bryan Chan, Anson Leung, and James Bergstra. Offline-to-online reinforcement learning for image-based grasping with scarce demonstrations. *arXiv preprint arXiv:2410.14957*, 2024.

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*, 34:15084–15097, 2021.

Zihang Dai. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997. URL `https://api.semanticscholar.org/CorpusID:1915014`.

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

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2(3):6, 2022.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. Awac: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pp. 7487–7498. PMLR, 2020.

Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Memory gym: Partially observable challenges to memory-based agents in endless episodes. *arXiv preprint arXiv:2309.17207*, 2023.

Subhojeet Pramanik, Esraa Elelimy, Marlos C Machado, and Adam White. Recurrent linear transformers. *arXiv preprint arXiv:2310.15719*, 2023.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/radford21a.html`.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Yanchao Sun, Shuang Ma, Ratnesh Madaan, Rogerio Bonatti, Furong Huang, and Ashish Kapoor. Smart: Self-supervised multi-task pretraining with control transformers. *arXiv preprint arXiv:2301.09816*, 2023.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *Proceedings of the ... IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 10 2012. doi: 10.1109/IROS.2012.6386109.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Kerong Wang, Hanye Zhao, Xufang Luo, Kan Ren, Weinan Zhang, and Dongsheng Li. Bootstrapped transformer for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:34748–34761, 2022.

Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *International Conference on Machine Learning*, pp. 38989–39007. PMLR, 2023.

Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *international conference on machine learning*, pp. 27042–27059. PMLR, 2022.

A Appendix

In Table 1 the first two columns describe MLP accelerator's parameters that were used during acceleration stage. The second two columns describe transformers parameters that were used during fine-tune stage. Since model parameters are fixed during both stages, this table also consists of all transformer's parameters that were used in acceleration stage too.

Table 1: Parameters that were used during both stages (wether it was acceleration stage or transformer fine-tuning stage).

	1st stage accelerator params		2nd stage transformer params	
Parameter	ManiSkill	MuJoCo	ManiSkill	MuJoCo
γ-discount	0.8	0.99	0.8	0.99
τ-soft update	0.01	0.005	0.01	0.005
Policy noise	0.2	0.2	0.2	0.2
Noise clip	0.5	0.5	0.5	0.5
Exploration noise	0.1	0.1	0.1	0.1
Batch size	600	256	256	256
Optimizer	Adam	Adam	Adam	Adam
Learning Rate	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Buffer size	0.05×10^{6}	0.5×10^{6}	0.01×10^{6}	0.1×10^{6}
Learning Starts	600	25000	0	0
Num Envs	50	1	50	1
Num layers	2	2	1	1
Num heads	-	-	2	2
Dim model	-	-	256	256
Dim feedforward	256	256	512	512
Dropout	-	-	0.05	0.05
Context len	-	-	3	3

B Supplementary materials for RQ1

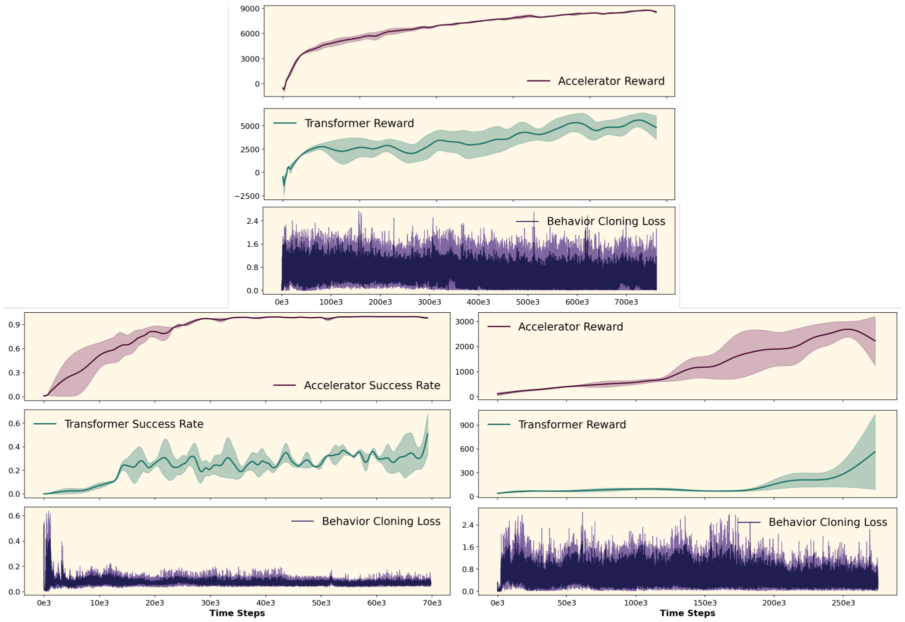

Figure 5: PushCube (bottom left), Hopper (bottom right) and HalfCheetah (top) acceleration progress.

C SUPPLEMENTARY MATERIALS FOR RQ2

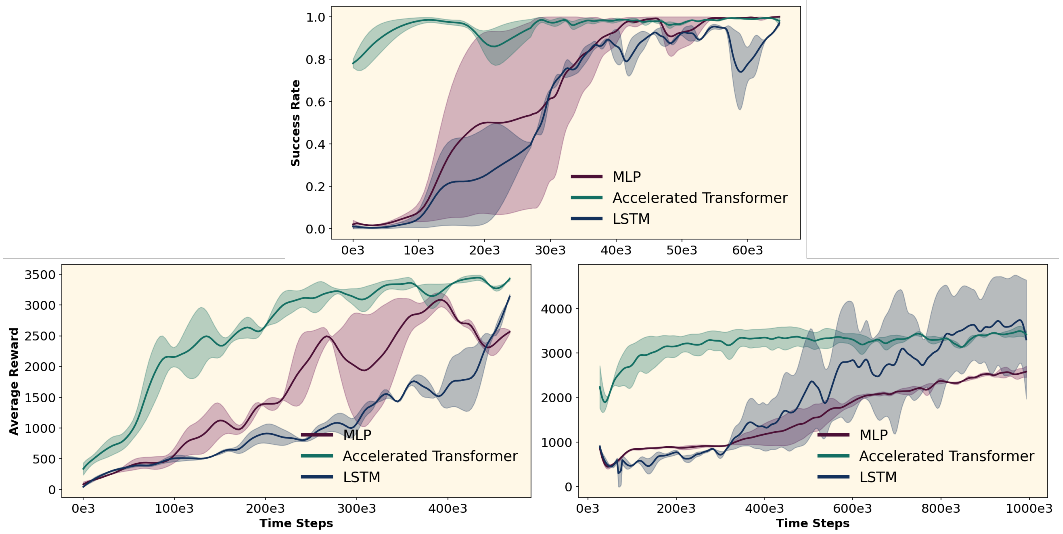

Figure 6: Online training progress. Success rate in PullCube (top) and average reward on Hopper (bottom left) and Ant (bottom right) tasks.

D SUPPLEMENTARY MATERIALS FOR RQ3

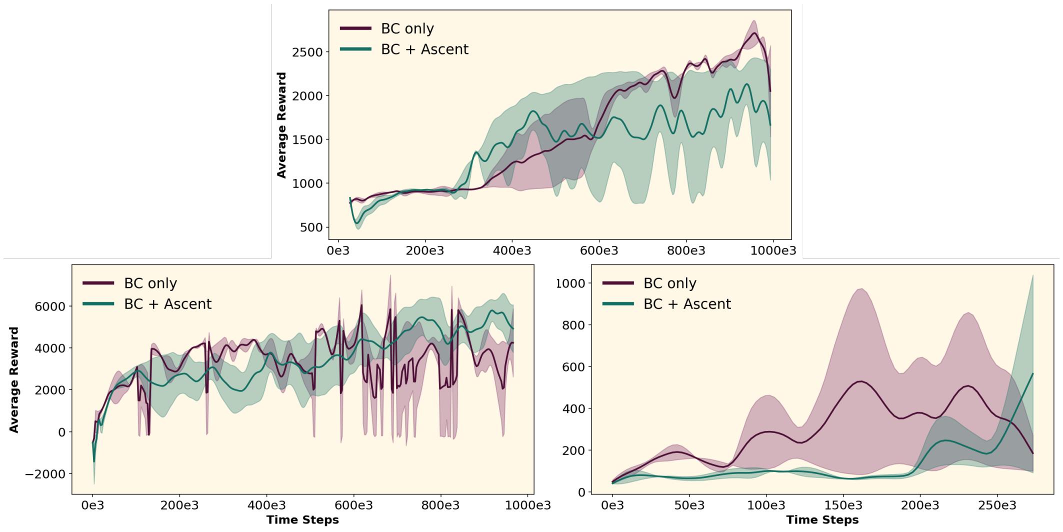

Figure 7: Acceleration progress with and without additional ascent. Ant (top), HalfCheetah (bottom left) and Hopper (bottom right) environments.