*Flexi*BiT: Flexible Inference in Sequential Decision Problems via Bidirectional Transformers

Anonymous authors
Paper under double-blind review

Abstract

Randomly masking sub-portions of sentences has been a very successful approach in training natural language processing models for a variety of tasks. In this work, we observe that the same idea also applies naturally to sequential decision making, where many traditional tasks like behavior cloning, offline RL, inverse dynamics, or planning correspond to different sequence maskings. We introduce the *Flexi*BiT framework, which enables to flexibly specify models which can be trained on many different sequential decision making tasks. Experimentally, we show that we can train a single *Flexi*BiT model to perform all tasks with performance similar to or better than specialized models, and that such performance can be further improved by fine-tuning this general model on the task of interest.

1 Introduction

Masked language modeling (Devlin et al., 2018) is a key technique in natural language processing (NLP). Under this paradigm, models are trained to predict randomly-masked subsets of tokens in a sequence. For example, during training, a BERT model might be asked to predict the missing words in the sentence "yesterday I ___ cooking a ___". Importantly, while unidirectional models like GPT (Radford et al., 2018) are trained to predict the next token conditioned only on the left context, bidirectional models trained on this objective learn to model both the left *and* right context to represent each word token. With these rich representations, the model can then be finetuned to excel on a variety of downstream tasks (Devlin et al., 2018).

Our work investigates how masked modeling can be a powerful idea in sequential decision problems. Consider a sequence of states s and actions a collected across T timesteps $s_1, a_1, \ldots, s_T, a_T$. If we consider each state and action as tokens of a sequence (analogous to words in NLP) and mask the last action, i.e., $(s_1, a_1, s_2, a_2, s_3, _)$, predicting the missing token a_3 amounts to a Behavior Cloning inference with two timesteps of history (Schaal, 1996; Argall et al., 2009). From this perspective, training a model to predict missing tokens from all maskings of the form $(s_1, a_1, \ldots, s_t, _)$ corresponds to training a Behavior Cloning (BC) model.

Similarly, many well-studied inference tasks can be expressed as a masking: goal or waypoint conditioned BC Rhinehart et al. (2019), offline reinforcement learning (RL) (Chen et al., 2021), forward or inverse dynamics prediction (Ha & Schmidhuber, 2018; Christiano et al., 2016), initial-state inference, and more. With this idea, we introduce the *Flexi*BiT framework, **Flexi**ble Inference in Sequential Decision Problems via **Bi**directional **T**ransformers. In this framework, a model can be trained to perform a variety of inference tasks in a unified way by expressing a task as a masking scheme and training on the masking schemes of interest. In contrast to standard approaches that train a different specialized model for each inference task, a single *Flexi*BiT model can perform a variety of tasks out-of-the-box.

We first test this framework in a gridworld navigation task with a *random* masking scheme, which corresponds to sampling all possible inference tasks at training time. We show how this scheme enables a single *Flexi*BiT model to condition on arbitrary subsets of states, actions, and rewards to perform a variety of inference tasks. We then systematically analyze how the masking schemes seen at training time affect downstream task performance. We find that, compared to specialized models that only train on the masking corresponding to the task of interest, a model trained on randomly

masked sequences outperforms on many tasks out-of-the-box. Fine-tuning on the task of interest further improves performance. We supplement these findings with preliminary experiments in continuous-control robotics environments which further support our claims. Our results suggest that expressing tasks as sequence maskings with the *Flexi*BiT framework may be a promising unifying approach to building general-purpose "multi-task" models.

2 Related Work

Transformer models. Transformer sequence models (Vaswani et al., 2017) have been successfully applied in other domains such as natural language processing (Devlin et al., 2018; Radford et al., 2018; Brown et al., 2020) and computer vision (Dosovitskiy et al., 2020; He et al., 2021). Using transformers in RL and sequential decision problems has proven difficult due to the instability of training (Parisotto et al., 2020), but recent work has investigated how to use transformers in model-based RL (Chen et al., 2022), motion forecasting (Ngiam et al., 2021), learning from demonstrations (Recchia, 2021), and tele-operation (Clever et al., 2021).

The utility of randomized masking. In addition to being used as one of the main training objectives for BERT (the "cloze task", Devlin et al. 2018), the flexibility afforded by randomized masking in bidirectional models has been utilized in other previous works applied to language (Ghazvininejad et al., 2019; Mansimov et al., 2019) and vision (Chang et al., 2022).

Sequential decision-making as sequence modeling. We are not the first to consider sequential decision problems as a sequence modeling problem. Chen et al. (2021) and Janner et al. (2021) focus on RL and show how one can use GPT-style (causally-masked) Transformer models to directly generate high-reward trajectories in an offline RL setting. Unlike this line of work, we focus on many tasks that a sequence modeling perspective enables one to do, rather than just offline-RL. Concurrent work to ours generalizes the left-to-right masking in the Transformer to condition on future trajectory information for tasks such as state marginal matching (Furuta et al., 2021) and multi-agent motion forecasting (Ngiam et al., 2021). In contrast to these works, we systematically investigate how a single bidirectional Transformer model can be trained to perform arbitrary downstream tasks (in more complex settings than simple motion forecasting—i.e., we also consider agent actions in addition to states). The main thing that sets us apart from all these works is a systematic view of all tasks that can be represented by this sequence-modeling perspective, and an detailed investigation of how different multi-task training regimes compare.

Self-supervised learning for sequential decision problems. Previous work on self-supervised learning is mostly focused on improving RL by training models on auxiliary objectives such as state dynamics prediction (Sekar et al., 2020) or intrinsic motivation (Pathak et al., 2017). Typically, to accomplish the tasks we consider, prior work relies on specialized models: for example, goal-conditioned imitation learning (Ding et al., 2019), RL(Kaelbling, 1993), waypoint-conditioning (Rhinehart et al., 2019), property-conditioning (Zhan et al., 2020; Furuta et al., 2021), or dynamics model learning (Ha & Schmidhuber, 2018; Christiano et al., 2016). In contrast, we demonstrate how sequence modeling can be a unifying framework for formulating and solving any inference task with a single model.

3 *Flexi*BiT

We introduce the *Flexi*BiT framework, first describing how common inference tasks in sequential decision problems can be formulated as masking schemes (Section 3.1), and then describing various possible training regimes (Section 3.2).

We model trajectories as sequences of states, actions, and return-to-go tokens:[1]

$$\tau = \{(s_1, a_1, \hat{R}_1), \ldots, (s_T, a_T, \hat{R}_T)\},$$

where the return-to-go $\hat{R}_t$ is the sum of rewards from timestep t to the end of the episode, $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$.

[1] While reward-to-go (or other trajectory statistics) are not necessary, we formulate the most general form to showcase how one can easily condition on additional properties of a trajectory if available. Using reward-to-go also enables us to baseline our method against previous offline-RL work (Chen et al., 2021).

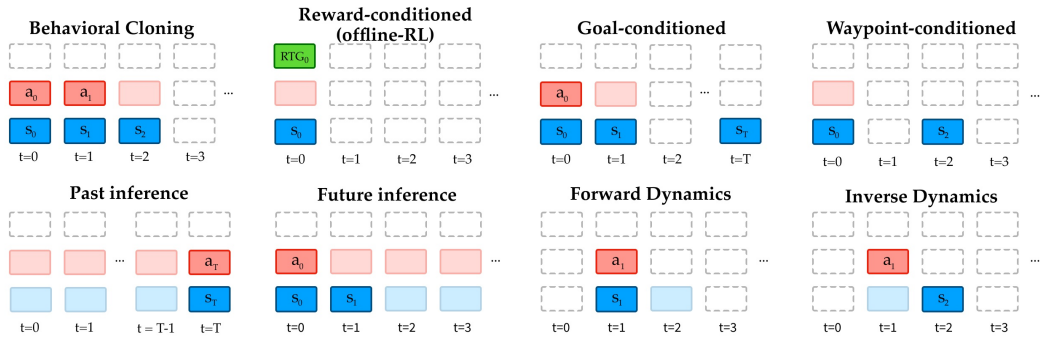

Figure 1: **A subset of the possible tasks that can be represented in the *Flexi*BiT framework.** For each task we show the inputs to the model and (with lower opacity) the predictions which we train the network to make for each input masking. For example, future inference tries to predict all future states and actions only conditional on initial states and actions. Here we display only one input masking scheme for each task, even though there might be multiple that are valid or necessary (e.g. BC will have up to T different masking schemes, one for each possible history length—although in practice one would generally use the model with a sliding window).

3.1 Tasks as Masking Schemes

In the *Flexi*BiT framework, we formulate tasks in sequential decision problems as an input masking schemes. Formally, a *masking scheme* is a function which randomly assigns masks to each token in trajectory snippets $\tau_{t:t+k}$ of length k drawn uniformly at random from a dataset. Such masking is applied both on the input space (determining what tokens are shown to the model for prediction) and on the output space (determining which outputs to predict). For example, conditioning on $s_{0:t}$ and $a_{0:t-1}$ and predicting a_t corresponds to one inference of the behavioral cloning (BC) task: $\mathbb{P}(a_t|s_{0:t}, a_{0:t-1})$.

In Figure 1, we illustrate how commonly-studied tasks such as BC, goal and waypoint conditioned imitation, offline RL (reward-conditioned imitation), and dynamics modeling can be unified under the representation of tasks as masking schemes. We describe the masking scheme for each of these tasks below.

For each trajectory snippet $\tau_{t:t+k}$ in each batch:

- **Behavioral Cloning (BC).** Select $i \in [0, k]$ uniformly. Feed $s_{t:i}, a_{t:i-1}$ to the network, with all other tokens masked out. Have the network only predict the next missing action a_i.
- **Goal-Conditioned imitation (GC).** Same as BC, but s_{t+k} is always unmasked.
- **Reward-Conditioned imitation (RC).** Same as BC, but return-to-go $\hat{R}_t$ is always unmasked.
- **Waypoint-Conditioned imitation (WPC).** Same as BC, but waypoints or subgoals $s_{t_1}, \ldots s_{t_n}$ are always unmasked.
- **Future inference (FUT).** Same as BC, but the model is trained to predict all future states and actions, rather than only the next missing action.
- **Past inference (PST).** Select $i \in [0, k]$ uniformly. Feed $s_{i:t+k}, a_{i:t+k}$ to the network, with all other tokens masked out. Have the network predict all previous states and actions $s_{t:i-1}, a_{t:i-1}$.
- **Forward dynamics (FWD).** Select $i \in [0, k-1]$ uniformly. Give the network the current state and action s_i, a_i, and have it predict the next state s_{i+1}. This enables to handle non-Markovian dynamics.
- **Past inference (INV).** Select $i \in [1, k]$ uniformly. Give the network the current state and previous action s_i, a_{i-1}, and have it predict the previous state s_{i-1}.
- **All the above (ALL).** Randomly select 1 of the above masking scheme functions and apply it to the current sequence. This is a simple way of performing multi-task training.

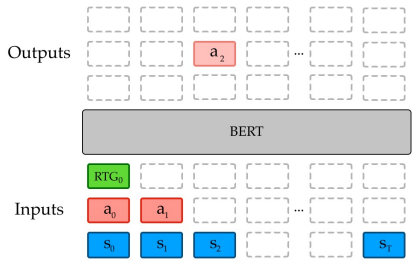

Figure 2: The *Flexi*BiT model takes in a snippet of a trajectory which is masked according to a masking scheme before inference time. For each input possible masking, there are (many) corresponding tasks of predicting the missing inputs. Above we show an input masking corresponding to conditioning on both reward *and* final (goal) state; we highlight the output corresponding to predicting the agent's next action, i.e. performing the inference $\mathbb{P}(a_2|s_{0:2,T}, a_{0:1}, RTG_0)$.

- **Random masking (RND).** Randomly select the masking probability for the sequence $p_{\text{mask}} \sim \text{Uniform}(0, 1)$. Mask each token independently with probability p_{mask}. More information on this input masking scheme in Appendix A.2. Have the model predict all masked tokens in the input (excepts returns-to-go). We treat the return-to-go $\hat{R}_0$ as a special token, masking it from the input with probability $p_{\text{mask}} = 0.5$. Randomly using the return-to-go in this fashion enables the model to perform both reward-conditioned and non-reward-conditioned tasks at inference time

3.2 Model Architecture & Training

As model architecture, we use the bidirectional transformer architecture from BERT (Devlin et al., 2018), consisting of stacked bidirectional transformer encoder (self-attention) layers, as shown in Figure 2. For more details on the model architecture, see Appendix A.1.

Given all of the possible masking schemes with which one could train *Flexi*BiT models, there will be many reasonable approaches to training and using such model at test. Below, we consider some, weighing their pros and cons:

0. **Training specialized models (e.g. BC)**: using a single *Flexi*BiT model for a single task. This involves training on only one of the single-task masking schemes from Section 3.1 (except ALL and RND, which are multi-task by nature). The only advantage this approach might have over conventional model types is the ease of use of tapping into the same architecture for every task instead of designing task-specific input pipelines; this is not our main focus;
1. **Training on multiple pre-specified useful tasks (e.g. ALL)**: with this setup, we aim to train a single *Flexi*BiT model that can perform well on all of its training tasks. We expect this setup to perform at least as well as custom models, as *Flexi*BiT may learn structure in the data from the other masking schemes that enables it to surpass the performance of the customized models;
2. **Training on random maskings (RND)**: with this approach, we aim to train a single model that performs well on any arbitrary sequence inference, without having to specify the tasks of interest before training. Moreover, this could potentially improve representations compared to 1). A disadvantage compared to 1) or even 0) is that depending on the model context length, the space of possible maskings could become too large for the model to be able to learn all tasks well;
3. **Training with methods 1 or 2, and then fine-tuning to a test-time task**: this should get the best of both worlds of multi and single-task models, taking advantage of the improved representations from multi-task training, and specializing the model parameters to the single task at hand (without having to share model capacity with other tasks). The cost would be greatly diminishing the multi-task test-time flexibility enabled by 1) or 2).

Hypotheses. Based on the above observations, we hypothesize that: **H1.** 3), which uses the full power of the framework, is often better than 0); **H2.** 1) and 2) are sometimes better than 0); **H3.**

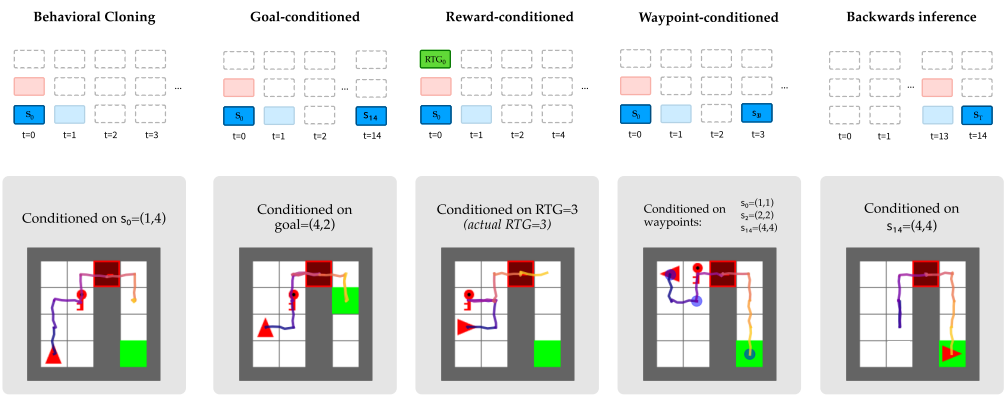

Figure 3: **A *Flexi*BiT model trained with random masking queried on various inference tasks. (1) Behavioral cloning:** generating an expert-like trajectory given an initial state. **(2) Goal-conditioned:** reaching an alternative goal. **(3) Reward-conditioned:** generating a trajectory that achieve a particular reward, e.g., by taking a suboptimal path that does not directly reach the goal. **(4) Waypoint-conditioned:** reaching specified waypoints (or subgoals) at particular timesteps, e.g. going down on the first timestep instead of immediately picking up the key. **(5) Backwards inference:** generating a likely *history* conditioned on a final state (by sampling actions and states backwards). Trajectories are shown with jitter for visual clarity.

when masking coverage of the desired tasks is sufficient, 2) is better than 1); **H4.** 3) is slightly better than 1) and 2);

In Section 4, we first show how training with a random masking scheme enables a single *Flexi*BiT model to perform arbitrary inference tasks at test-time on a simple navigation environment dataset, without the need for specialized output heads or training schemes that are customized for the downstream task. We then additionally show how on specific tasks of interest, training with random maskings can achieve comparable or better performance to specialized models or multi-task models (trained on a short list of possibly relevant tasks), despite rarely seeing that particular task at training time. After training on the masked prediction task, we show how *Flexi*BiT can additionally be fine-tuned on the specific task of interest, which we show further improves performance.

In Section 5, we present preliminary experimental results from more complex robotics environments, showcasing that performing random-masking training and fine-tuning can obtain comparable or better reward performance in the behavior cloning task.

4 MINIGRID EXPERIMENTS

4.1 EXPERIMENTAL SETUP

To showcase the flexibility of our model, we use a grid-like navigation environment in which we might be interested in many different types of inferences. We use the minigrid framework (Chevalier-Boisvert et al., 2018) with a custom DoorKey environment. The environment is a fully observable 4-by-4 gridworld in which the agent should move to a fixed goal location which is behind a locked door. To be able to pass the locked door, the agent must first pick up a key. Both the agent and key are initialized at random locations in each episode outside the locked room with the goal. The agent receives a reward of 1 for each timestep it moves closer to the goal, -1 if it moves away from the goal, and 0 otherwise. We pre-train *Flexi*BiT as described below on training trajectories of sequence length $T = 10$ from a noisy-rational agent (Ziebart et al., 2008) that takes the optimal action with probability $a \sim p(a) \propto \exp(C(a))$ where $C(a) = 1$ if the distance to the current goal (key or final goal) decreases, -1 if it increases, and 0 otherwise. More detailed information about the environment is in Appendix A.3.

4.2 ONE MODEL TO RULE THEM ALL

We first showcase the out-of-the-box flexibility of a *Flexi*BiT model when trained with random masking. By querying the model in a variety of ways, we show qualitatively in Figure 3 that it is

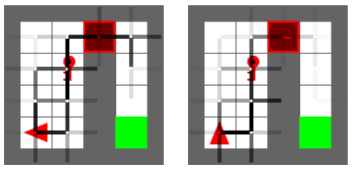

Figure 4: Distribution of states and actions for trajectories in the validation set, vs. trajectories sampled from the model, conditioned on the initial agent position (1,4) and key position (2,2).

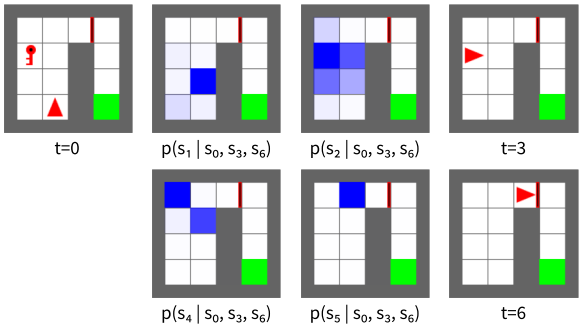

Figure 5: Predicted state distributions, conditioned on states at $t = 0, 3, 6$.

able to perform common tasks. Unless otherwise indicated, we take the highest probability action from the model $a_t = \arg\max_{a'_t} p(a'_t \mid s_0, a_0, \ldots, s_t)$, and then query the environment dynamics for the next state s_{t+1}. In some tasks, trajectories rolled out from the model may not always be consistent with the observed factors (e.g. the model's highest probability trajectory may not reach the conditioned waypoints). In these cases, one could use a search procedure such as beam search to expand several trajectory hypotheses, eliminating the ones that are inconsistent with observed factors, although we did not find this to be necessary in the gridworld examples.

Note that with random masking, seeing the exact masking corresponding to a particular task at training time is exceedingly rare; there are $2^T \cdot 2^T \cdot 2$ possible state, action, and reward maskings for a sequence of length T (in these experiments $T = 10$, resulting in about two million possible maskings). This suggests that the model is capable of generalizing across input-masking schemes. As seen in the backwards inference task (where we also query the model for state predictions), we can also effectively use *Flexi*BiT as an inverse dynamics model; in our framework, this is simply yet another task captured by the corresponding masking scheme.

4.3 Future State Predictions

In Figure 3 we show how we can iteratively use the randomized-masking *Flexi*BiT model to sample trajectories in the environment. We visualize the distribution of states and actions for trajectories sampled from the model, conditioned on the initial state (behavioral cloning). As seen in Figure 4, the model learns to match the underlying distribution of trajectories in (heldout) data.

Uses of the random-masking-trained *Flexi*BiT model are not limited to the *iterative sampling* of new trajectories (requesting inferences about the agent's next action). One can also request inferences for states and actions further into the future: e.g., "where will the agent be *in 3 timesteps*?". Given a *fixed* initial set of observed states, we visualize the distribution of predicted states at each timestep in Figure 5. Since we do not roll out actions, querying the model for the predicted state distribution at a particular timestep marginalizes over missing actions; for example, $p(s_1 \mid s_0, s_3, s_6)$ models the possibility that the agent chooses either up or left as the first action. Looking at these predictions, we see that most of the state predictions made by the model seem plausible, suggesting that it is able to appropriately leverage it's knowledge of dynamics and usual agent behavior. In particular, it correctly models that the agent has equal probability of going up and right at $t = 3$ (leading it to the distribution over states at $t = 4$), and that the agent must be at position (2, 1) at $t = 5$ in order to reach the door at $t = 6$.

4.4 How do different training regimes compare?

If we care about a single task (e.g. goal-conditioned imitation), should we train a model simply on that task? Or can there be advantages to training a general model first, and then fine-tuning it to the task of interest? To shed light on these questions we compiled a heatmap (Figure 6) which reports validation losses across a variety of models and evaluations.

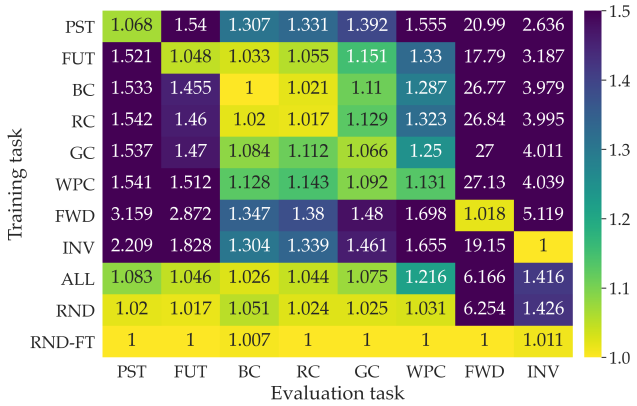

Figure 6: **Task-specific validation losses (normalized column-wise).** Each row corresponds to the performance of a single model evaluated in various ways, with the exception of the RND-FT row—for which each cell is finetuned on the respective evaluation task. Loss values are averaged across 3 seeds and then divided by the smallest value in each column. Thus, for each evaluation task (i.e., column), the best method has value 1; a value of 1.5 corresponds to a loss that is 50% higher than the best model in the column. See Appendix A.5 for unnormalized values.

Single-task models do well on their task and poorly on others. Firstly, notice as a sanity check that all entries on the diagonal (the specialized models, trained and evaluated on a single task) are generally quite good (their rescaled loss values are quite low). Also, as they are *Flexi*BiT models, they can also be evaluated on other tasks that they weren't trained on (just by changing the input maskings): generally, as one would expect, these single tasks model tend to suffer quite a bit of degradation in performance when evaluating on previously unseen evaluation tasks.

Multi-task models perform as well as specialized ones (and sometimes better). Turning our attention to the third and second to last rows, we see multi-task models: for both models, we have a single model that is able to reasonably perform across all the evaluation tasks considered. In the case of the randomized masking, there are two notable takeaways: 1) it leads to lower loss values across almost all evaluation tasks relative to the ALL model, giving credence to the idea that it might be beneficial to broaden the number of tasks trained on dramatically—even if one is interested in a single task (this supports **H3**); 2) often the random-masking *Flexi*BiT model is able (even before fine-tuning) to obtain better performance than the specialized models (in 4/8 of the tasks)—supporting **H2**; 3) we see that this proportion increases to 6/8 tasks after additionally fine-tuning this general *Flexi*BiT model to the specific task of interest (supporting **H1** and **H4**).

Forward and inverse dynamics. The forward and inverse dynamics evaluations of multi-task models stick out as being particularly poor performance relative to specialized and fine-tuned models. This is because this simple environment has deterministic dynamics, meaning that with sufficient data (as in this case), it's essentially impossible to overfit ("overfitting is fitting"). Training a model for long enough exclusively on forward dynamics gives loss values quite close to zero. This means that any multi-task model which is trading off between that task and others might fail to overfit quite as dramatically (a gap which can be easily made up for during fine-tuning). In this sense, it might be more appropriate to consider the heatmap without these tasks. Still, we wanted to demonstrate how well our model could perform these tasks, if necessary.

5 MUJOCO EXPERIMENTS

Next, we report results for preliminary experiments on the HalfCheetah, Hopper, and Walker Mujoco environments (Todorov et al., 2012). We use the D4RL *Mujoco medium expert datasets* (Fu et al., 2020) which we downsample to speed up experimentation.

Task setup. We evaluate models exclusively on the behavior cloning task, considering cumulative reward in evaluation rollouts as the performance metric (unlike in the minigrid experiments). We

again compare the performance of *Flexi*BiT when using several training regimes introduced in Section 3.1, namely, BC, RND, and RND+FT. Additionally, we baseline with a simple Neural Network model trained to perform behavior cloning (NN-BC), and with a Decision Transformer (Chen et al., 2021) trained without rewards-to-go[2] (DT-BC). See appendix A.4 for more details about the setup.

Results. Across all environments considered, the best reward is obtained by first training a *Flexi*BiT model with random masking and then fine-tuning it to the BC task. This further supports **H1** and **H4**. Among the specialized models, *Flexi*BiT performs surprisingly well, generally outperforming its GPT-based and NN-based counterparts. *Flexi*BiT trained on RND alone lags behind in performance relative to the specialized BC model, but performs similarly to NN-BC and DT-BC (partially supporting **H2**).

Environment	**NN-BC**	**DT-BC**	***Flexi*BiT-BC**	***Flexi*BiT-RND**	***Flexi*BiT-RND+FT**
Hopper	1579±290	1472±162	1534±100	1413±215	**1610±63**
HalfCheetah	3616±181	3781±178	4286±131	3451±435	**4466±117**
Walker	2487±234	2626±158	2811±170	2565±125	**2969±113**

Table 1: We see that across the various Mujoco environments, the best reward is obtained by training a *Flexi*BiT model with random masking and then fine-tuning it to BC. We report the standard errors over reward evaluations performed by 3 different models (with different seeds).

6 Limitations and Future Work

Property-conditioning. The choice of representing trajectories as states, actions, and returns-to-go is not constrained by the model. One could use any property of the environment or the agent as additional tokens (or set of tokens), enabling to perform property-conditioned inferences, or infer such properties from test-time trajectory snippets. Reward-conditioning is just one example of such property-conditioning.

Short context lengths. One limitation in our experimentation are the relatively short context lengths used. This is because 1) the architecture we use requires memory quadratic in the sequence length, and 2) the number of possible random maskings is exponential in the sequence length. 1) could be addressed by using efficient transformers (Wang et al., 2020; Jaegle et al., 2021); 2) could be addressed by designing masking schemes tailed to specific test-time tasks (see Appendix A.2), or using a principled masking scheme (Levine et al., 2020).

Other future work. An important axis of improvement for our results would be to further scale the domains we consider to more complex ones. Another exciting direction for future work is trying to see whether the benefits obtained from random masking (or even multi-task masking) would also apply to Bayes Networks more generally; alternatively, even trivially extending the approach to multi-agent settings (for which token-stacking could prove more valuable), could enable many interesting masking-enabled queries Ngiam et al. (2021).

7 Conclusion

In this work we propose *Flexi*BiT, a framework for flexibly defining and training models which: **1)** are naturally able to represent any inference task and support multi-task training in sequential decision problems, **2)** match or surpass the performance of specialized models after multi-task pre-training, and almost always surpasses them after fine-tuning. We believe our approach provides an elegant conceptual formulation and exciting experimental results which warrant further investigation by the research community.

[2]This is equivalent to using a GPT-like model to perform behavior cloning—used in the original paper for their Atari BC baselining.

REFERENCES

Brenna D. Argall, Sonia Chernova, Manuela M. Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics Auton. Syst.*, 57(5):469–483, 2009. doi: 10.1016/j.robot.2008.10.024. URL https://doi.org/10.1016/j.robot.2008.10.024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. Maskgit: Masked generative image transformer. *CoRR*, abs/2202.04200, 2022. URL https://arxiv.org/abs/2202.04200.

Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. Transdreamer: Reinforcement learning with transformer world models. *CoRR*, abs/2202.09481, 2022. URL https://arxiv.org/abs/2202.09481.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv:2106.01345 [cs]*, June 2021. URL http://arxiv.org/abs/2106.01345. arXiv: 2106.01345.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model. *arXiv:1610.03518 [cs]*, October 2016. URL http://arxiv.org/abs/1610.03518. arXiv: 1610.03518.

Henry M. Clever, Ankur Handa, Hammad Mazhar, Kevin Parker, Omer Shapira, Qian Wan, Yashraj S. Narang, Iretiayo Akinola, Maya Cakmak, and Dieter Fox. Assistive tele-op: Leveraging transformers to collect robotic task demonstrations. *CoRR*, abs/2112.05129, 2021. URL https://arxiv.org/abs/2112.05129.

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.

Yiming Ding, Carlos Florensa, Mariano Phielipp, and Pieter Abbeel. Goal-conditioned imitation learning. *arXiv preprint arXiv:1906.05838*, 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

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

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 6111–6120. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1633. URL https://doi.org/10.18653/v1/D19-1633.

David Ha and Jürgen Schmidhuber. World Models. *arXiv:1803.10122 [cs, stat]*, March 2018. doi: 10.5281/zenodo.1207631. URL http://arxiv.org/abs/1803.10122. arXiv: 1803.10122.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.

Michael Janner, Qiyang Li, and Sergey Levine. Reinforcement Learning as One Big Sequence Modeling Problem. *arXiv:2106.02039 [cs]*, June 2021. URL http://arxiv.org/abs/2106.02039. arXiv: 2106.02039.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

Yoav Levine, Barak Lenz, Opher Lieber, Omri Abend, Kevin Leyton-Brown, Moshe Tennenholtz, and Yoav Shoham. Pmi-masking: Principled masking of correlated spans. *arXiv preprint arXiv:2010.01825*, 2020.

Elman Mansimov, Alex Wang, and Kyunghyun Cho. A generalized framework of sequence generation with application to undirected sequence models. *CoRR*, abs/1905.12790, 2019. URL http://arxiv.org/abs/1905.12790.

Jiquan Ngiam, Benjamin Caine, Vijay Vasudevan, Zhengdong Zhang, Hao-Tien Lewis Chiang, Jeffrey Ling, Rebecca Roelofs, Alex Bewley, Chenxi Liu, Ashish Venugopal, David Weiss, Ben Sapp, Zhifeng Chen, and Jonathon Shlens. Scene Transformer: A unified multi-task model for behavior prediction and planning. *arXiv:2106.08417 [cs]*, June 2021. URL http://arxiv.org/abs/2106.08417. arXiv: 2106.08417.

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.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 488–489, 2017.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training, 2018.

Gabriel Recchia. Teaching autoregressive language models complex tasks by demonstration. *arXiv preprint arXiv:2109.02102*, 2021.

Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. PRECOG: PREdiction Conditioned On Goals in Visual Multi-Agent Settings. *arXiv:1905.01296 [cs, stat]*, September 2019. URL http://arxiv.org/abs/1905.01296. arXiv: 1905.01296.

Stefan Schaal. Learning from demonstration. In Michael Mozer, Michael I. Jordan, and Thomas Petsche (eds.), *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pp. 1040–1046. MIT Press, 1996. URL http://papers.nips.cc/paper/1224-learning-from-demonstration.

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, P. Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. *ArXiv*, abs/2005.05960, 2020.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Eric Zhan, Albert Tseng, Yisong Yue, Adith Swaminathan, and Matthew Hausknecht. Learning calibratable policies using programmatic style-consistency. In *International Conference on Machine Learning*, pp. 11001–11011. PMLR, 2020.

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum Entropy Inverse Reinforcement Learning. pp. 6, 2008.

A APPENDIX

A.1 MODEL DETAILS

Input stacking. An important hyperparameter for transformer models is what dimension to use self-attention over. Previous work applies it across states, actions, and rewards as separate tokens (or even individual state and action dimensions) Chen et al. (2021); Janner et al. (2021); this can increase the effective sequence length that we would need to input a trajectory snippet of length k: for example if treating states, actions, and rewards separately (have self-attention act on each independently), the sequence length would be $3k$. While this is not an issue for the short context windows we use in our experiments, this seems wasteful: the main bottleneck for transformer models is usually the computational cost of self-attention, which scales quadratically in the sequence length. To obviate this problem, we stack states, actions, and rewards for each timestep, treating them as a single input. This way, we are making self-attention happen only across time, and reducing the sequence length required to k. This also seems like a potentially advantageous inductive bias for improving performance. Though we did not test this systematically, preliminary experiments did show that input stacking had a positive effect on performance.

Return-to-go conditioning. Unlike previous work that considers return-to-go conditioning (Chen et al., 2021; Janner et al., 2021), we don't provide the model with many return-to-go tokens (one for each timestep). Providing just the first token should be sufficient for the model to interpret the return-to-go request (as, if necessary, the model can compute the remaining return to go in later steps). We found this did not negatively impact performance.

A.2 RANDOM MASKING SCHEME DETAILS

Given the correspondence of maskings and tasks, we found that a masking scheme that uniformly masks each token with a fixed probability p (as common in NLP) to be a naive strategy, particularly for longer sequence lengths. With this approach, the number of tokens masked will be distributed as $n \sim Binomial(N, p)$ (where N is the number of tokens). This means that it will be highly unlikely to see either 0 or N tokens masked – even though there are many meaningful tasks that require most tokens to be masked (past prediction) or unmaksed (behavior cloning with full history). We fix this problem by using the randomized masking scheme described in Section 3.1, which we found to perform much better.

A.3 MINIGRID DETAILS

Below we delineate some more details about our custom DoorKey minigrid environment.

State and action space, and more details on environment dynamics

The state and action spaces are both represented as discrete inputs: there are 4 actions, corresponding to the 4 possible movement directions (up, right, down, left); taking each action will move the agent in the corresponding direction unless 1) the agent is facing a wall, 2) the agent is facing the locked door without a key. Stepping on the key location tile picks up the key. The state is represented as two one-hot encoded position vectors—the agent position and the key position (which is equivalent to the agent position once the key has been picked up). Both agent and key position have 16 possible values, some of which are never seen in the data (e.g. the agent position coinciding with a wall location). Together, such vectors are sufficient to have full observability for the task—as seeing the key location coincide with the agent location informs the model that the agent is holding the key, and if the agent is holding the key it can open the locked door. Once the agent is holding the key, whether the door is open or closed is irrelevant.

Having the states and actions be discrete enables all model predictions to be done on a discrete space—which is particularly convenient as it enables the trained models to output any distribution over predicted states and actions, which can be easily visualized such as in Figure 5.

***Flexi*BiT models hyperparameters for DoorKey experiments**

As the DoorKey environment have discrete actions and states, we use the softmax-cross-entropy loss over all predictions.

For the trained models we use 3 stacked encoder layers, 8 attention heads, 128 hidden dimension size, and 128 embedding dimension size. We train with a learning rate of 10^{-4}, and but scale the state and action losses by a factor of 0.2. We train with the torch implementation of the Adam optimizer. We use batch size 100. We train until we reach the lowest validation loss, or up to a maximum of 6000 epochs.

When fine-tuning, we reduce the learning rate to 5×10^{-5} and train for 500–1500 epochs depending on the task.

A.4 MUJOCO EXPERIMENT DETAILS

Datasets

The D4RL datasets are very large, consisting of thousands of trajectories with hundreds of steps—which is far more data than is necessary to perform BC successfully (as they were designed for offline-RL experimentation). We found that training models on even 5% of the data provided barely reduced the BC performance of the various models—and that all models were performing almost at the level of the expert that generated the data. Both due to this and to speed up experimentation, we decided to reduce the dataset size dramatically to: 10 trajectories for hopper, 5 for walker2d, and 11 for halfcheetah (respectively 0.5%, 1%, and 0.5% of the original dataset sizes).

Model hyperparameters for Mujoco experiments

As the Mujoco environments have continuous actions and states, we use a L2 loss over all predictions. Mujoco trajectories for the environments we considered have up to 1000 timesteps. In our experiments we use a model context length of 5, which acts as a "sliding window" over the trajectories in the data.

For the trained models we use 3 stacked encoder layers, 8 attention heads, 128 hidden dimension size, and 128 embedding dimension size (except on halfcheetah, which benefitted from a 256 embedding size). We train with a learning rate of 5×10^{-4}. We train with the torch implementation of the Adam optimizer. We use batch size 100. We train for 10000 epochs and save the model which performs best in reward evaluations (we verify that all runs were converged at 10000 epochs—so as to remove the confounder of additional training in the fine-tuned case).

When fine-tuning, we reduce the learning rate to 10^{-5} and train for another 10000 epochs, again saving the best performing model. Similar hyperparameters are used across all models: NN, DT, and *Flexi*BiT.

A.5 MORE VALIDATION-LOSS RESULTS IN THE MINIGRID DOMAIN.

We report below more validation-loss results from the minigrid experiments, varying the amount of data used to train all models. We try dataset sizes of 50, 500, and 1000. We see that notwithstanding the differences in dataset size, the trends and relative orderings of performance between models tend to stay the same.

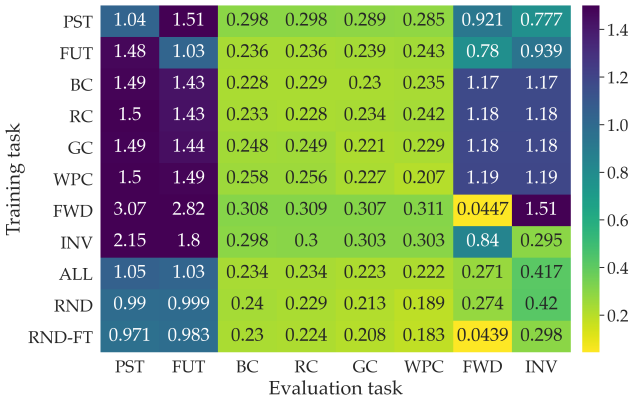

Figure 7: The raw loss values corresponding to Figure 6.

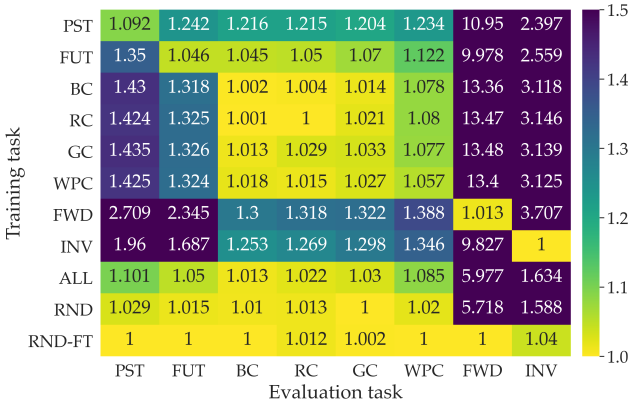

Figure 8: Figure 6 when using a dataset of 50 trajectories instead of 500. We see that most of the trends remain the same.

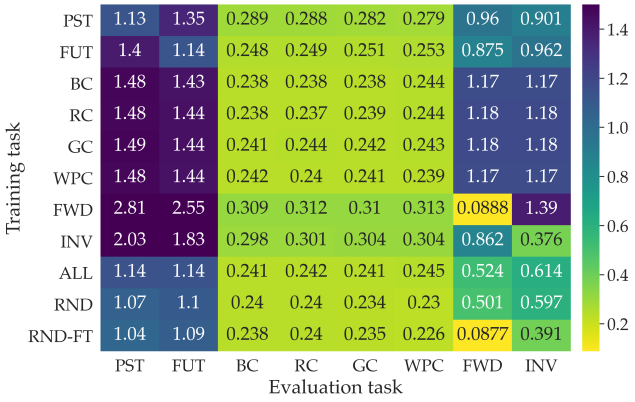

Figure 9: The raw loss values corresponding to Figure 8.

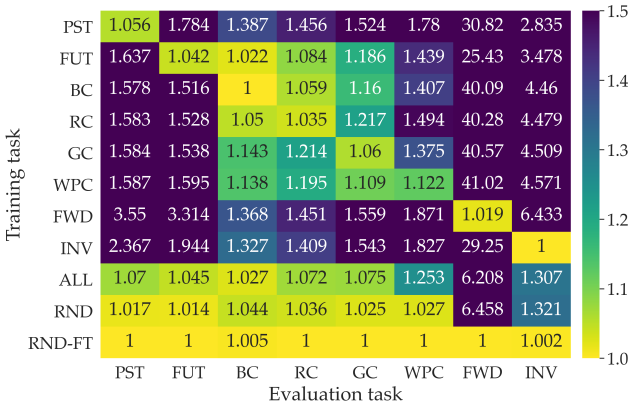

Figure 10: Figure 6 when using a dataset of 1000 trajectories instead of 500. We see that most of the trends remain the same.

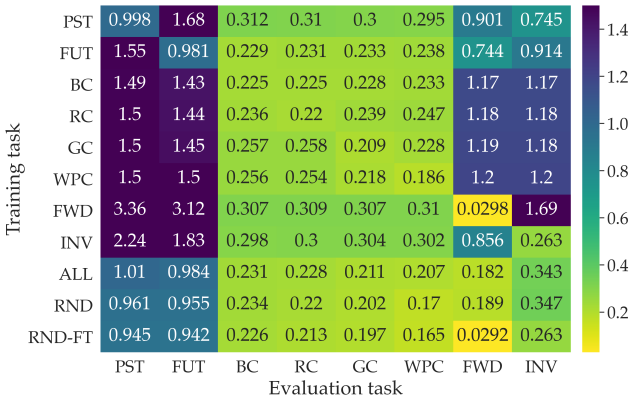

Figure 11: The raw loss values corresponding to Figure 10.