Decision Transformer: Reinforcement Learning via Sequence Modeling

Lili Chen[*,1], **Kevin Lu**[*,1], **Aravind Rajeswaran**[2], **Kimin Lee**[1],
Aditya Grover[2], **Michael Laskin**[1], **Pieter Abbeel**[1], **Aravind Srinivas**[†,1], **Igor Mordatch**[†,3]
[*]equal contribution [†]equal advising
[1]UC Berkeley [2]Facebook AI Research [3]Google Brain
{lilichen, kzl}@berkeley.edu

CS 6784 Advanced Topics in Machine Learning: ML for Feedback Systems
Paper Presentation
Menghan Xu
Nov. 4, 2025

Problem Statement

Treat sequential decision making as RL and solve it via Transformers

What we Do?

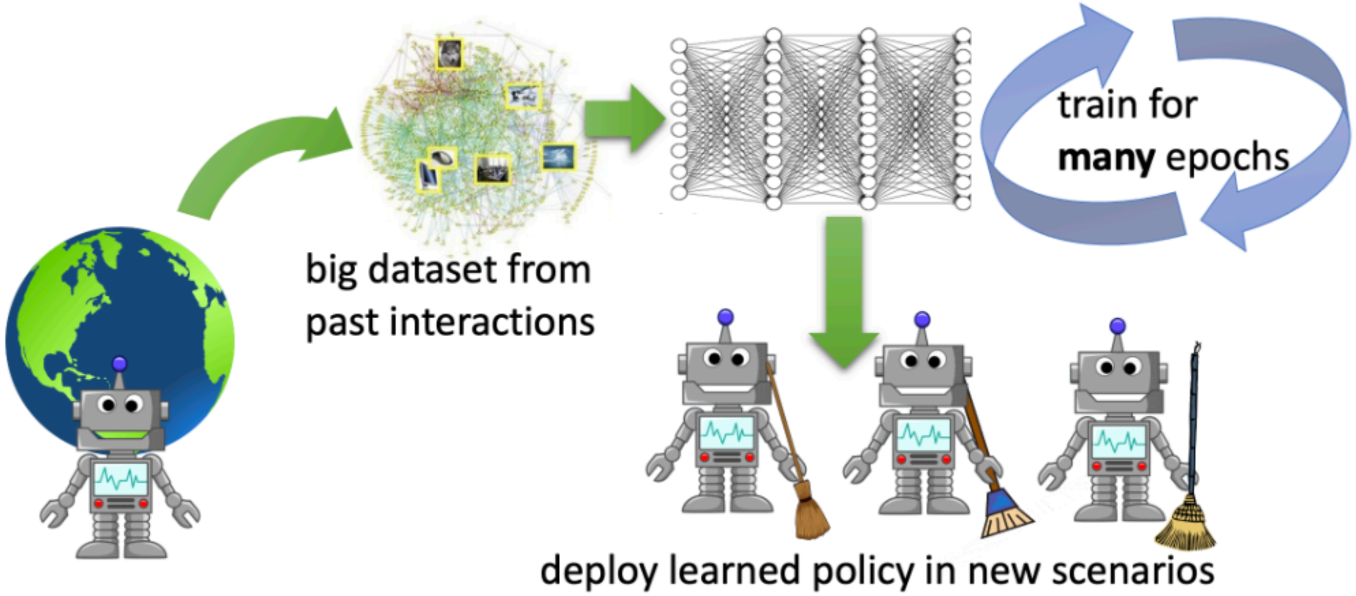
- how to formalize?
- how to apply Transformer?

Why we do it?

- does it actually work?
- why can this approach succeed?

What we do?
Problem set up - offline RL

offline reinforcement learning



trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$

Markov Decision Process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \mathcal{R})$$

- $s_t \in \mathcal{S}$: state
- $a_t \in \mathcal{A}$: action
- $P(s'|s, a)$: transition dynamics
- $r_t = \mathcal{R}(s_t, a_t)$: reward

Don't learn this(i.e. model free)

$$\tau = (s_0, a_0, r_0, \ldots, s_T, a_T, r_T) \qquad \text{(trajectory)}$$

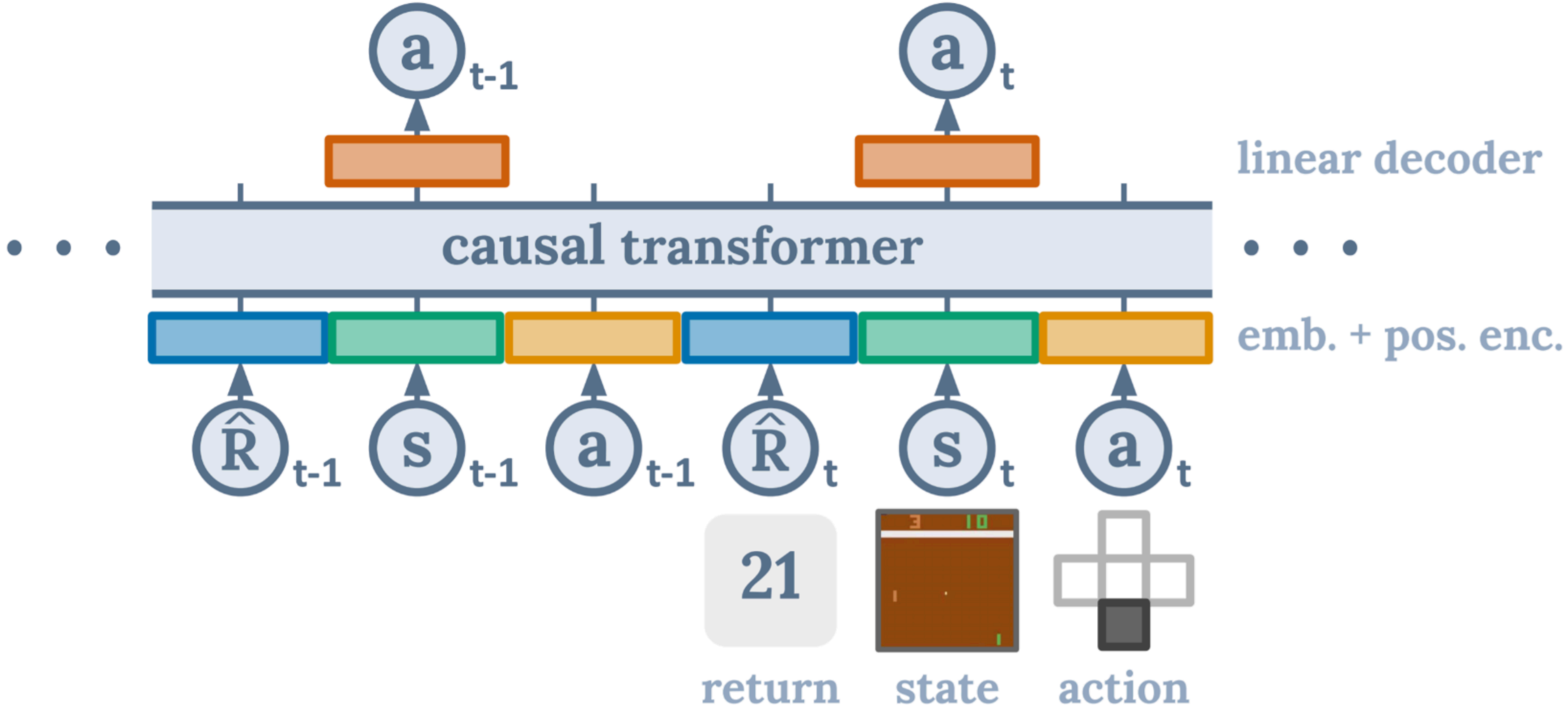
$$R = \sum_{t=0}^{T} r_t \qquad \text{(total return)}$$

$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'} \qquad \text{(return-to-go)}$$

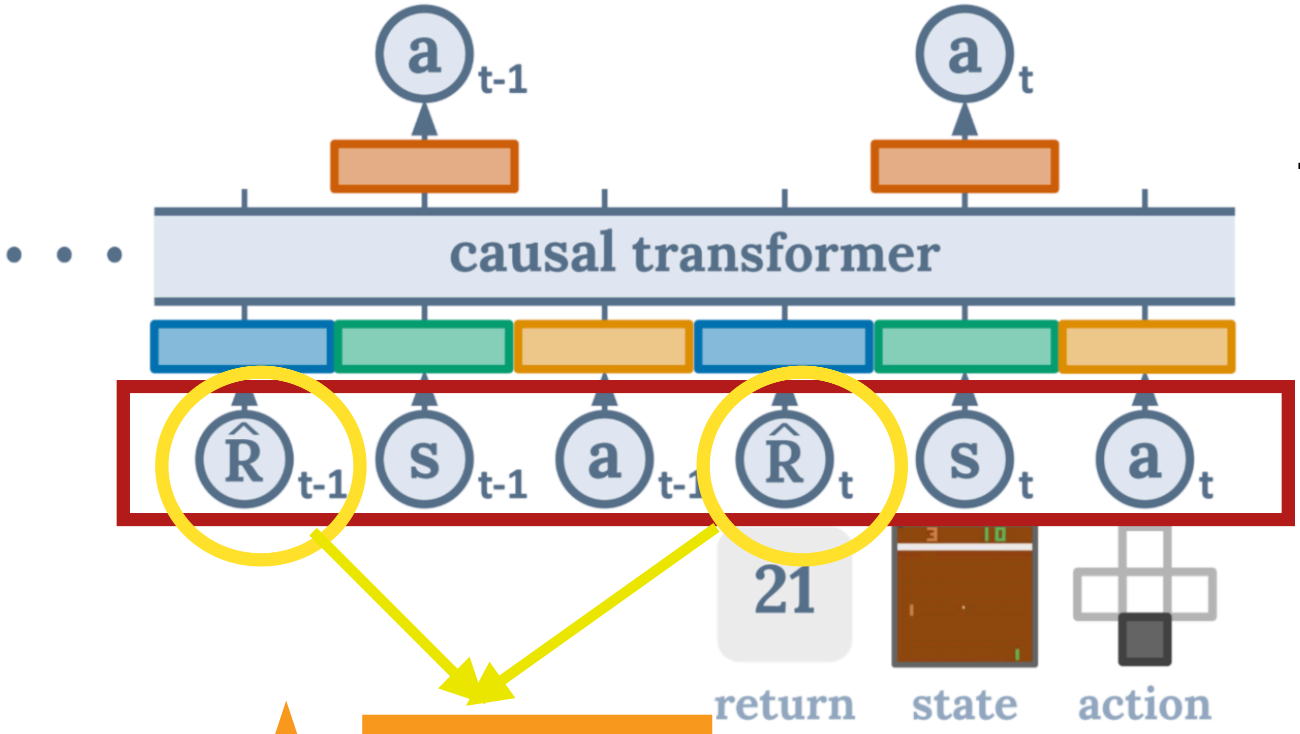
Goal:

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} r_t\right]$$

What we do?
Overall Architecture



What we do?



Key Desiderata

Trajectory representation:

$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'} \qquad \text{(return-to-go)}$$

$$\tau = \left(\boxed{\widehat{R}_1, s_1}, a_1, \widehat{R}_2, s_2, a_2, \ldots, \widehat{R}_T, s_T, a_T\right)$$

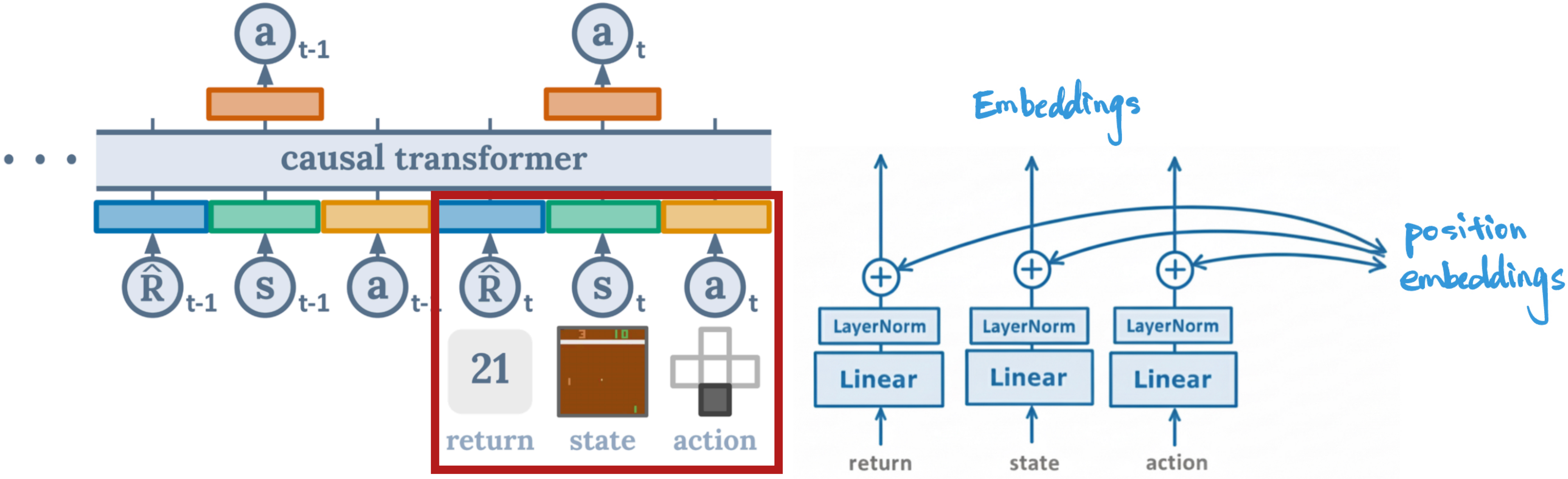
Desired performance

Initial state

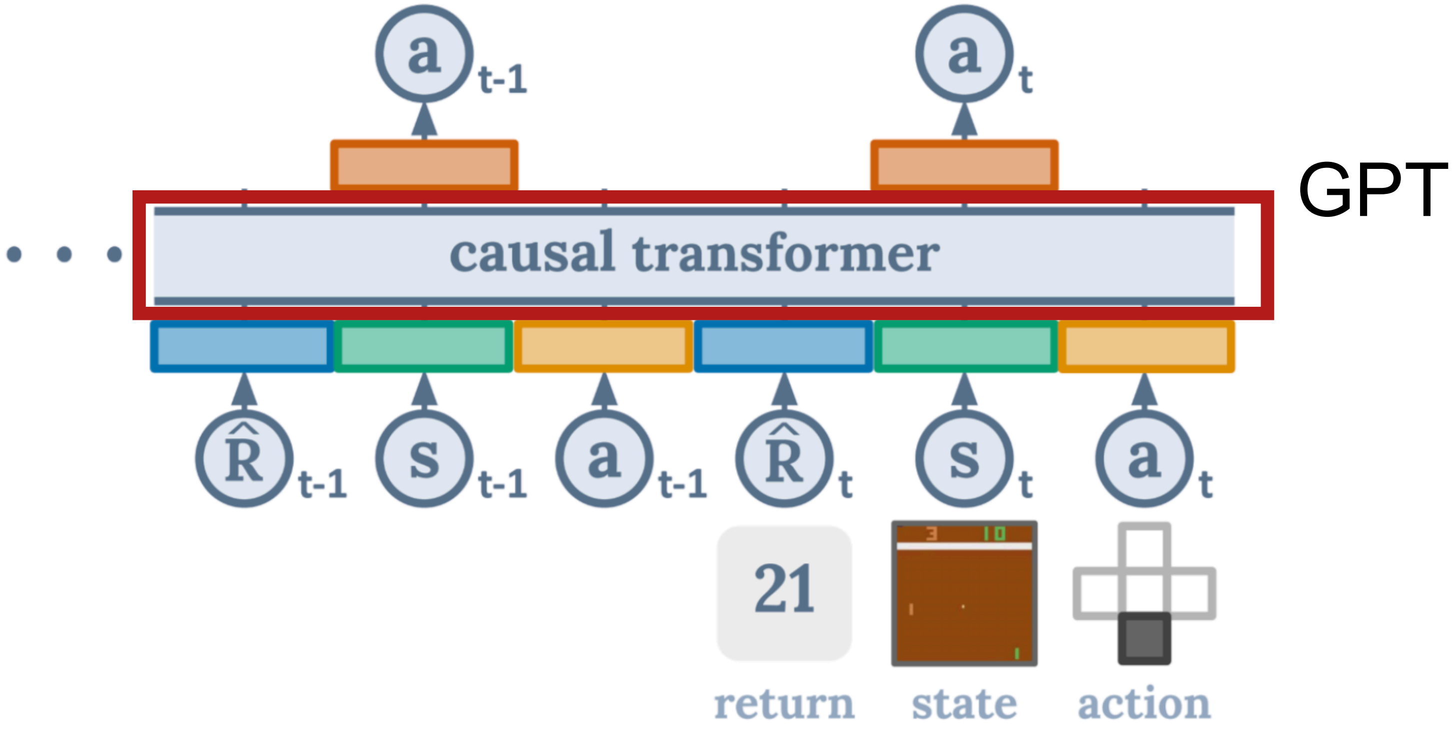
At test time

$$\hat{R}_{t+1} = \hat{R}_t - r_t$$

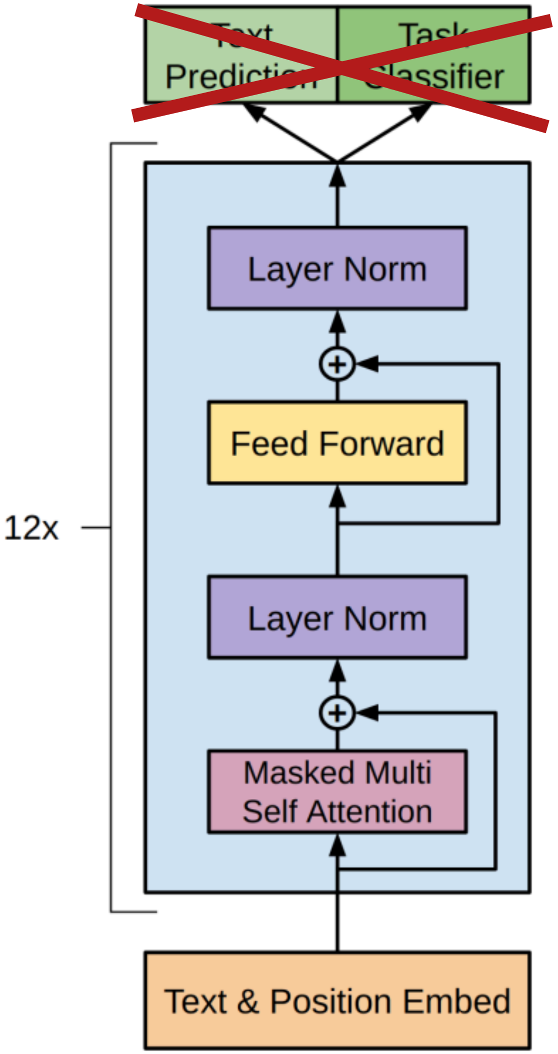
What we do?


a_{t-1}
a_t
causal transformer
$\hat{R}_{t-1}$
s_{t-1}
a_{t-1}
$\hat{R}_t$
s_t
a_t
21
return
state
action
Embeddings
position embeddings
LayerNorm
Linear
return
state
action

What we do?

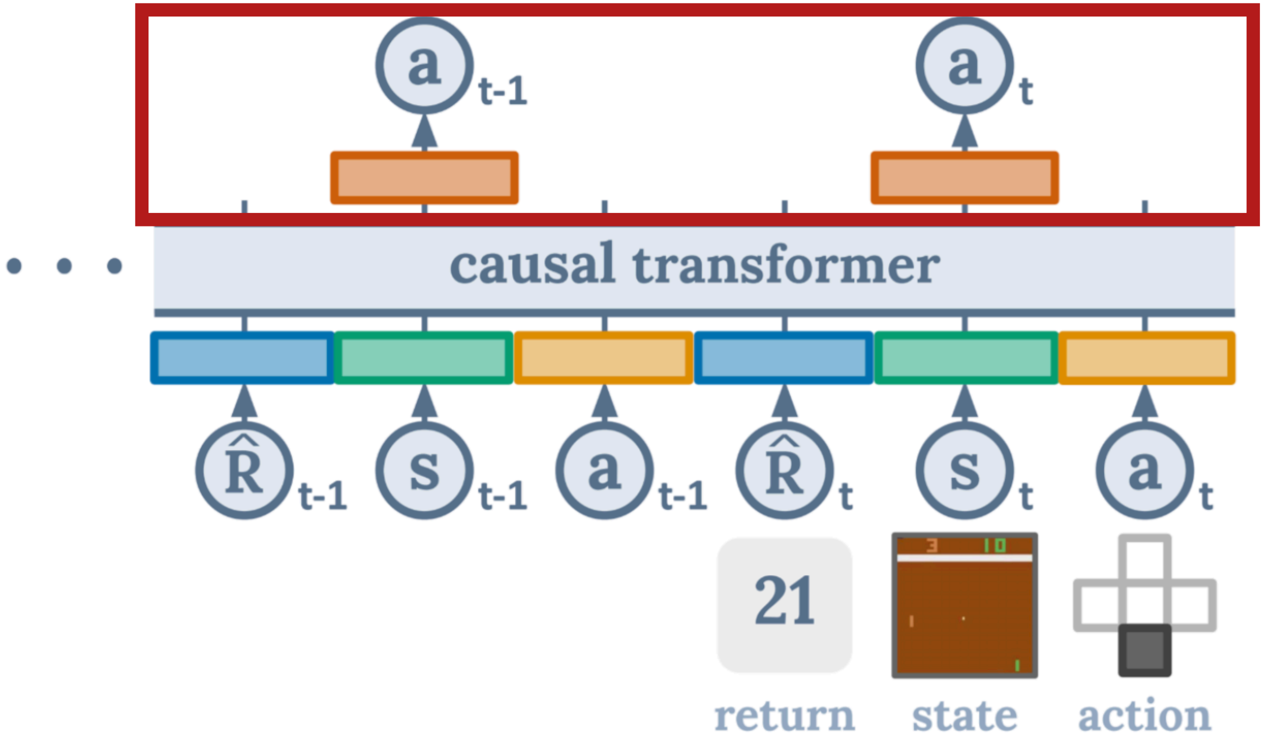


a t-1
a t
causal transformer
GPT
R̂ t-1
s t-1
a t-1
R̂ t
s t
a t
21
return
state
action



~~Text Prediction~~
~~Task Classifier~~
Layer Norm
Feed Forward
Layer Norm
Masked Multi Self Attention
12x
Text & Position Embed

What we do?



Only select hidden states corresponding to **action tokens**

$$h_t = \text{Transformer}(\hat{R}_{0:t}, s_{0:t}, a_{0:t-1})_t$$

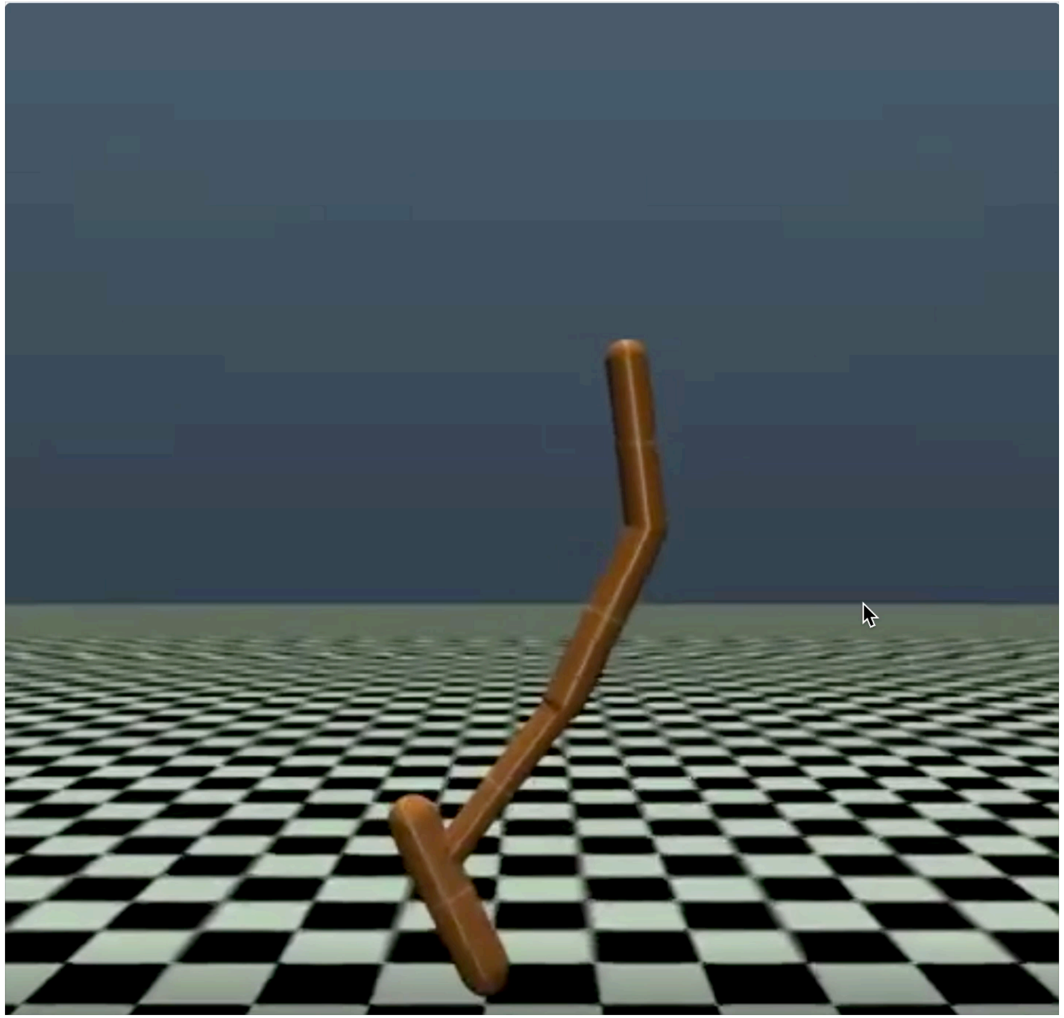
Continuous actions: MSE loss

$$\mathcal{L}_{\text{cont}} = \frac{1}{K}\sum_{t=1}^{K} \|a_t - \hat{a}_t\|_2^2$$

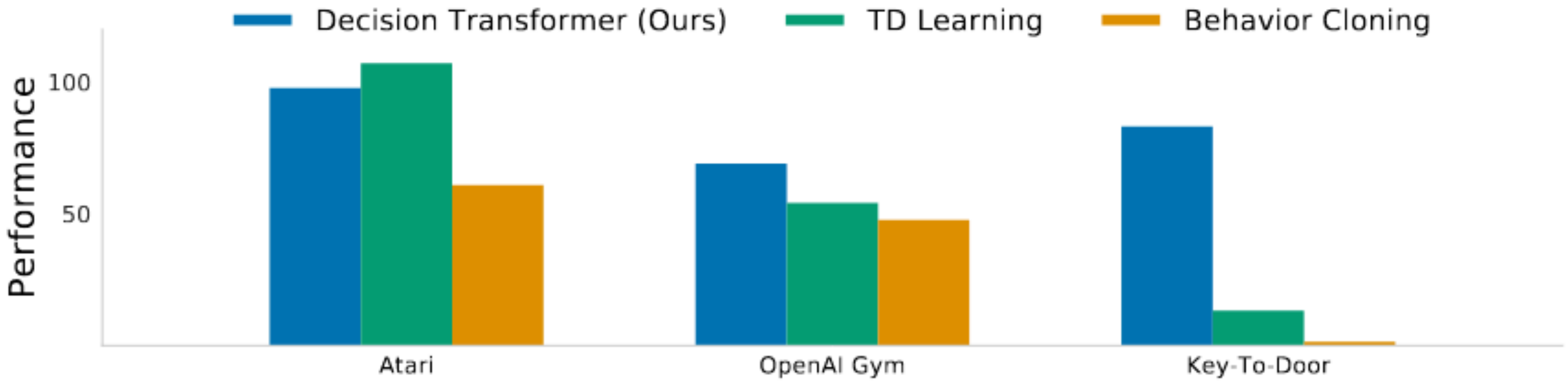
Discrete actions: cross-entropy loss

$$\mathcal{L}_{\text{disc}} = \frac{1}{K}\sum_{t=1}^{K} -\log \pi_\theta(a_t \mid h_t)$$

Why we do it? Experiments



Why we do it?
Experiments



Why we do it?
Experiments

Atari

Game	DT (Ours)	CQL	QR-DQN	REM	BC
		TD Learning			
Breakout	**267.5 ± 97.5**	211.1	17.1	8.9	138.9 ± 61.7
Qbert	15.4 ± 11.4	**104.2**	0.0	0.0	17.3 ± 14.7
Pong	106.1 ± 8.1	**111.9**	18.0	0.5	85.2 ± 20.0
Seaquest	**2.5 ± 0.4**	1.7	0.4	0.7	2.1 ± 0.3

OpenAI Gym

Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	**86.8 ± 1.3**	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	**111.0**	96.3	0.8	27.1	79.6
Medium-Expert	Walker	**108.1 ± 0.2**	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	**89.1 ± 1.3**	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	**67.6 ± 1.0**	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	**51.2 ± 3.4**	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	**82.7 ± 7.0**	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	**66.6 ± 3.0**	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	**18.0 ± 2.4**	**19.0**	-	-	-	5.4
Average (Without Reacher)		**74.7**	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		**69.2**	54.2	-	-	-	47.7

Why we do it?

Does Decision Transformer perform behavior cloning on a subset of the data?

Dataset	Environment	DT (Ours)	10%BC	25%BC	40%BC	100%BC	CQL
Medium	HalfCheetah	42.6 ± 0.1	42.9	43.0	43.1	43.1	**44.4**
Medium	Hopper	$\mathbf{67.6 \pm 1.0}$	65.9	65.2	65.3	63.9	58.0
Medium	Walker	74.0 ± 1.4	78.8	**80.9**	78.8	77.3	79.2
Medium	Reacher	51.2 ± 3.4	51.0	48.9	58.2	**58.4**	26.0
Medium-Replay	HalfCheetah	36.6 ± 0.8	40.8	40.9	41.1	4.3	**46.2**
Medium-Replay	Hopper	$\mathbf{82.7 \pm 7.0}$	70.6	58.6	31.0	27.6	48.6
Medium-Replay	Walker	66.6 ± 3.0	**70.4**	67.8	67.2	36.9	26.7
Medium-Replay	Reacher	18.0 ± 2.4	**33.1**	16.2	10.7	5.4	19.0
Average		56.1	**56.7**	52.7	49.4	39.5	43.5

Why we do it?

Does Decision Transformer perform behavior cloning on a subset of the data?

low data regime - Atari

use 1% of a replay buffer as the dataset, so then %BC is weak

Game	DT (Ours)	10%BC	25%BC	40%BC	100%BC
Breakout	**267.5 ± 97.5**	28.5 ± 8.2	73.5 ± 6.4	108.2 ± 67.5	138.9 ± 61.7
Qbert	15.4 ± 11.4	6.6 ± 1.7	16.0 ± 13.8	11.8 ± 5.8	**17.3 ± 14.7**
Pong	**106.1 ± 8.1**	2.5 ± 0.2	13.3 ± 2.7	72.7 ± 13.3	85.2 ± 20.0
Seaquest	**2.5 ± 0.4**	1.1 ± 0.2	1.1 ± 0.2	1.6 ± 0.4	2.1 ± 0.3

Takeaway: DT is not just cloning the best trajectories

Why we do it?
How well does Decision Transformer model the distribution of returns?

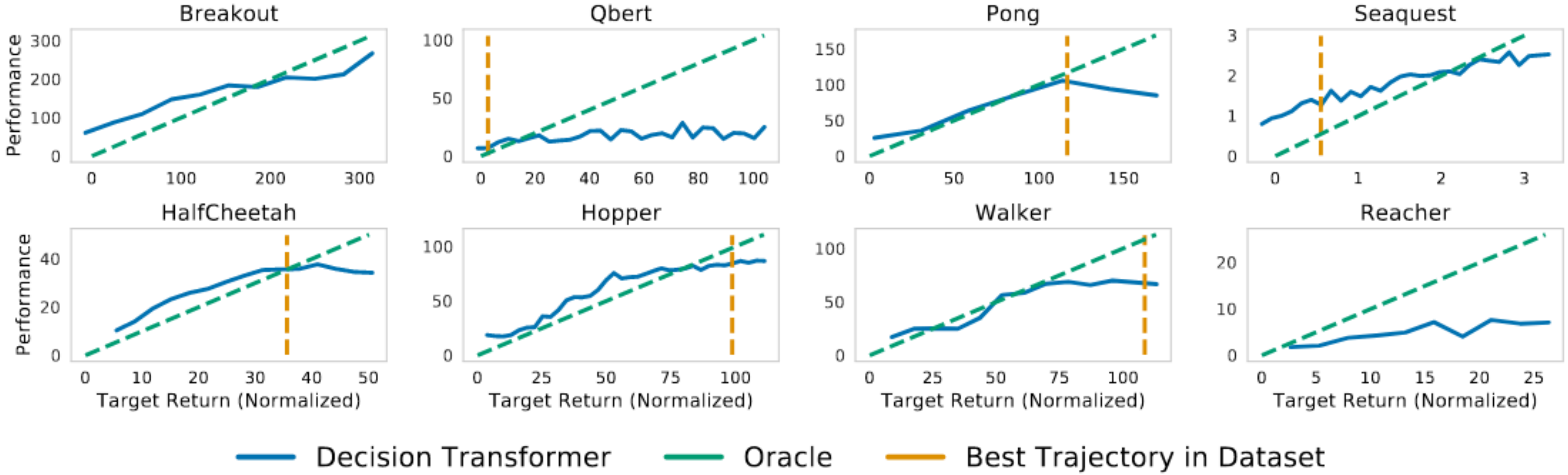


Figure 4: Sampled (evaluation) returns accumulated by Decision Transformer when conditioned on the specified target (desired) returns. **Top:** Atari. **Bottom:** D4RL medium-replay datasets.

Takeaway: DT successfully conditions on return-to-go and can adjust behavior across return levels, sometimes even extrapolating beyond the dataset

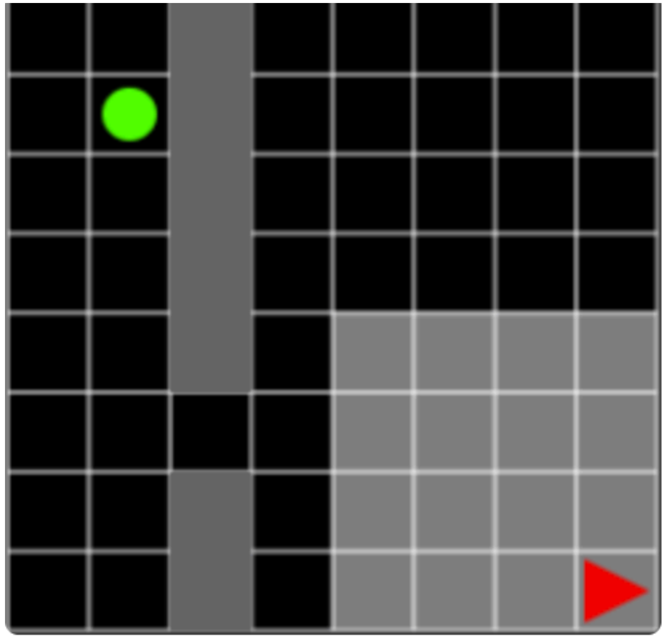
Why we do it?
What is the benefit of using a longer context length?

Game	DT (Ours)	DT with no context ($K = 1$)
Breakout	$\mathbf{267.5 \pm 97.5}$	73.9 ± 10
Qbert	$\mathbf{15.1 \pm 11.4}$	13.6 ± 11.3
Pong	$\mathbf{106.1 \pm 8.1}$	2.5 ± 0.2
Seaquest	$\mathbf{2.5 \pm 0.4}$	0.6 ± 0.1

Table 5: Ablation on context length. Decision Transformer (DT) performs better when using a longer context length ($K = 50$ for Pong, $K = 30$ for others).

Takeaway: DT really uses the past

Why we do it?
In sparse reward setting



a variant to the key to door

3 phases:

1. the agent is placed in a room with a key
2. the agent is placed in an empty room
3. the agent is placed in a room with a door

3 problems in this setting:

1. Does DT perform effective long-term credit assignment? YES
2. Can DT be accurate critics in sparse reward settings? YES
3. Does DT perform well in sparse reward settings? YES

Why we do it? In sparse reward setting
Does Decision Transformer perform effective long-term credit assignment? YES!

Dataset	DT (Ours)	CQL	BC	%BC	Random
1K Random Trajectories	**71.8%**	13.1%	1.4%	69.9%	3.1%
10K Random Trajectories	94.6%	13.3%	1.6%	**95.1%**	3.1%

Table 6: Success rate for Key-to-Door environment. Methods using hindsight (Decision Transformer, %BC) can learn successful policies, while TD learning struggles to perform credit assignment.

Why we do it? In sparse reward setting Can transformers be accurate critics? YES!

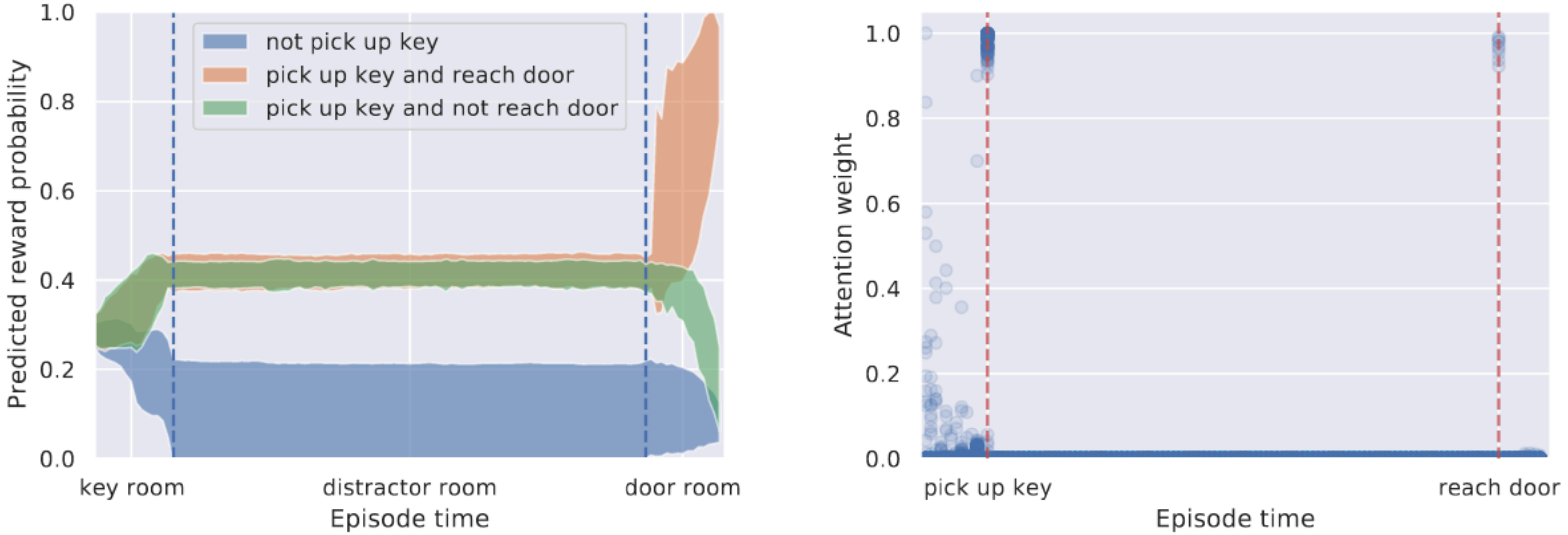


Figure 5: **Left:** Averages of running return probabilities predicted by the transformer model for three types of episode outcomes. **Right:** Transformer attention weights from all timesteps superimposed for a particular successful episode. The model attends to steps near pivotal events in the episode, such as picking up the key and reaching the door.

Why we do it? In sparse reward setting Does Decision Transformer perform well? YES!

		Delayed (Sparse)		Agnostic		Original (Dense)	
Dataset	**Environment**	**DT (Ours)**	**CQL**	**BC**	**%BC**	**DT (Ours)**	**CQL**
Medium-Expert	Hopper	$\mathbf{107.3 \pm 3.5}$	9.0	59.9	102.6	107.6	111.0
Medium	Hopper	60.7 ± 4.5	5.2	63.9	**65.9**	67.6	58.0
Medium-Replay	Hopper	$\mathbf{78.5 \pm 3.7}$	2.0	27.6	70.6	82.7	48.6

Table 7: Results for D4RL datasets with delayed (sparse) reward. Decision Transformer (DT) and imitation learning are minimally affected by the removal of dense rewards, while CQL fails.

Why we do it?
Why does Decision Transformer avoid the need for value pessimism or behavior regularization?

Traditional offline RL (e.g., TD-learning) learns an **approximate value function** and **optimizes a policy** on top of it

But optimizing an imperfect value function can **amplify errors** → **unstable policy improvement**

So prior methods require **policy regularization / conservatism** (e.g., CQL) to stay close to the dataset and avoid value overestimation

Decision Transformer does NOT optimize a value function

- It treats RL as **sequence modeling**, using supervised learning
- Therefore, it **does not need explicit regularization** to remain stable

Why we do it?
How can Decision Transformer benefit online RL regimes?

Strong offline behavior prior
Learns diverse high-quality trajectories from offline data
→ good starting policy for online learning

Likelihood-based training transfers well
Sequence modeling objective has shown smooth offline→online adaptation

Acts as a "memory engine"
Remembers good behaviors and can regenerate them during online training

Supports broad behavior exploration
When combined with exploration methods (e.g., Go-Explore),
enables generating diverse and high-return trajectories

POST DT work

- Online Decision Transformer https://arxiv.org/abs/2202.05607
- Value-Guided/Q-Guided DT https://arxiv.org/abs/2209.03993
- Diffussion policies/Multimodal Agents https://arxiv.org/abs/2409.00588

Question?

References

- https://arxiv.org/abs/2106.01345
- https://sites.google.com/berkeley.edu/decision-transformer
- https://openreview.net/forum?id=a7APmM4B9d
- https://bair.berkeley.edu/blog/2020/12/07/offline/
- https://greentec.github.io/reinforcement-learning-fifth-en/
- https://arxiv.org/pdf/2006.04779
- https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- https://github.com/souradipp76/SeqModRL?tab=readme-ov-file