Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Transformer-based Algorithmics

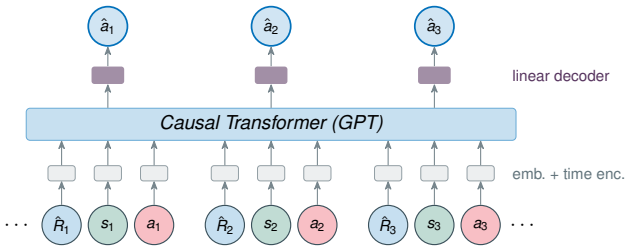
Malte Luttermann – Institute for Humanities-Centered Artificial Intelligence (CHAI)

July 15, 2026

Version from July 13, 2026

Topics for Today

- Reinforcement learning (RL)
- Reinforcement learning *as sequence modeling*
- Decision transformer (DT)

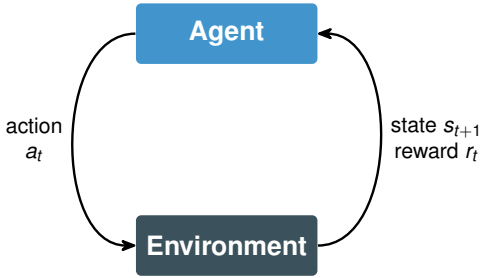


one transformer, three token types

Reinforcement Learning

An **agent** acts in an **environment**, collecting a trajectory of states, actions and rewards.

- Goal: maximise the total return $\sum_t r_t$.



$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$$

Two Recipes for the Same Goal

Classic RL

- Estimate values via *Bellman equation*.
- Derive a policy: $\pi(s) = \arg\max_a Q(s, a)$.
- Iterative, unstable, needs careful tuning.

Decision Transformer

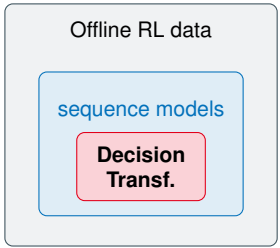
- Treat the trajectory as a *sequence of tokens*.
- Just predict the next action — supervised learning.
- No value function, no Bellman equation.

Central question

Can we get good decisions *purely* by modeling sequences — the same way GPT models text?

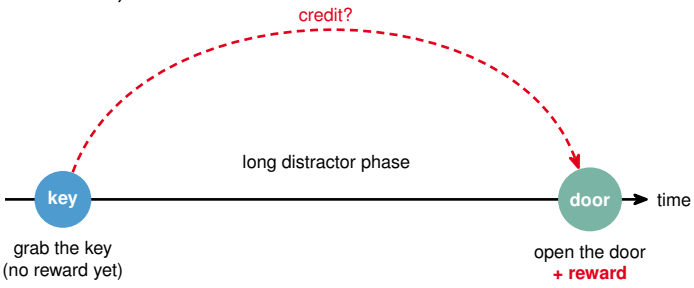
Where the Decision Transformer Sits

- ✓ **Offline:** learns from a *fixed* dataset — no new interaction.
- ✓ **Model-free:** no transition model $P(s' \mid s, a)$.
- ✓ **Supervised:** predicts actions, no Bellman updates.
- ✓ **Sequence model:** a GPT over $(\hat{R}, s, a)$ tokens.



Problem 1 — Credit Assignment

Which early action caused a reward that only arrives much later? (Or: Which actions deserve credit?)



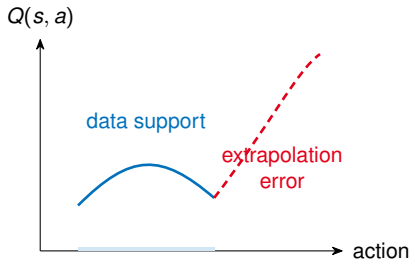
- Bellman updates pass credit *one step at a time* — slow over long horizons.
- A Transformer attends *directly* from the door back to the key.

Problem 2 — Distribution Shift (Offline RL)

With a fixed dataset, value-based methods query Q at *unseen* actions.

- $\arg\max_a Q(s, a)$ picks rarely-seen (s, a).
- Errors there are never corrected $\Rightarrow$ overestimation.
- Wildly inaccurate predictions for these out-of-distribution (OOD) states.

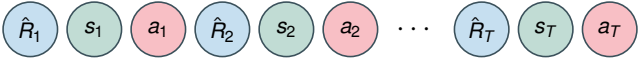
Decision transformer avoids this: it never bootstraps a Q-value.



Trajectories as Sequences: Returns-to-Go

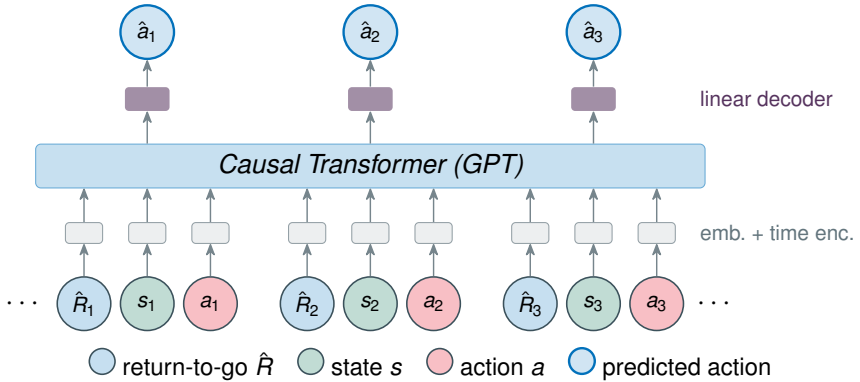
Replace each reward by the **return-to-go** — the total future reward from that step on:

$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'}.$$



- The first token $\hat{R}_1$ is the *desired* return we condition on.
- At each step the model sees “how much reward is still wanted” — and picks an action accordingly.

The Decision Transformer Training Pipeline

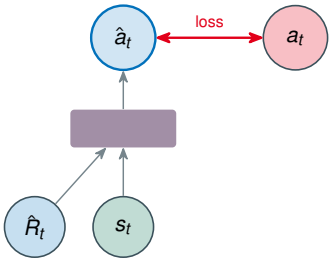


Each token is linearly embedded, a per-timestep encoding is added, and a GPT with *causal* masking predicts the action at every state position.

Training — Just Supervised Learning

Sample a length-K chunk of a trajectory, feed the $(\hat{R}, s, a)$ tokens, predict the actions.

$$\mathcal{L} = \frac{1}{K} \sum_t \left\| a_t - \hat{a}_t \right\|^2$$



Inference — Return-Conditioned Generation

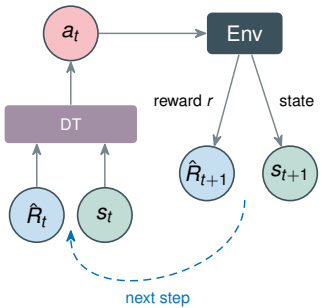
Tell the model what you want.

(1) Set a target return $\hat{R}_0$.

(2) Model outputs an action; run it in the environment.

(3) Observe reward r and decrement: $\hat{R}_{t+1} = \hat{R}_t - r$.

(4) Append $(\hat{R}_{t+1}, s_{t+1})$, repeat.



$\hat{R}_{t+1} = \hat{R}_t - r$; state and reward from Env

Exercise 1 — Returns-to-Go

A short episode produced the reward sequence

$$r_1 = 0, \quad r_2 = 2, \quad r_3 = 0, \quad r_4 = 5.$$

Tasks.

(1) Compute the returns-to-go $\hat{R}_1, \ldots, \hat{R}_4$.

(2) Write out the token sequence the DT would see (states s_t, actions a_t abstract).

(3) The dataset's best episode scores 7. If you condition on $\hat{R}_1 = 10$, is that guaranteed to work? Why / why not?

Exercise 1 — Solution

(1) Returns-to-go ($\hat{R}_t = \sum_{t' \geq t} r_{t'}$):

$$\hat{R}_1 = 7, \quad \hat{R}_2 = 7, \quad \hat{R}_3 = 5, \quad \hat{R}_4 = 5.$$

(2) Token sequence.

$$(\, 7, s_1, a_1,\ 7, s_2, a_2,\ 5, s_3, a_3,\ 5, s_4, a_4 \,)$$

(3) Conditioning on 10**.**

- Not guaranteed: 10 lies *outside* the training returns.
- DT can sometimes *extrapolate* a bit beyond the best seen return. . .
- . . . but only if the data contains the necessary sub-behaviours to stitch together.

Exercise 2 — The Inference Loop

You condition the DT on a target return $\hat{R}_0 = 8$ and roll out. The environment returns rewards

$$r_1 = 3, \qquad r_2 = 1, \qquad r_3 = 2.$$

Tasks.

(1) Give the returns-to-go tokens $\hat{R}_1, \hat{R}_2, \hat{R}_3$ fed back at each step.

(2) What does $\hat{R}_3$ intuitively represent?

(3) Suppose after three steps the return-to-go hits 0 but the episode is not over. What might go wrong?

Exercise 2 — Solution

(1) Decrement $\hat{R}_t = \hat{R}_{t-1} - r_t$:

$$\hat{R}_1 = 5, \qquad \hat{R}_2 = 4, \qquad \hat{R}_3 = 2.$$

(2) Meaning of $\hat{R}_3$. The reward still "owed" to reach the original target — the model keeps steering toward it.

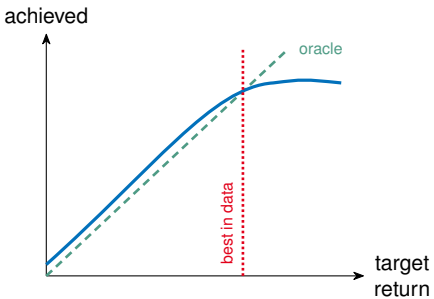
(3) Return-to-go $= 0$ early.

- The model was told "no reward left to collect".
- It may switch to passive / low-value behaviour.
- Take-away: the *target return matters* — set it high enough for the whole episode.

Conditioning Actually Works

Ask for a return, get roughly that return.

- Along the diagonal = the model obeys the command.
- On some tasks DT even *extrapolates* beyond the best trajectory in the data.
- Eventually it plateaus — can't invent behaviour that isn't there.



Discussion — Promise vs. Caveats

Why it's appealing.

- Dead simple: one supervised loss.
- No instability from bootstrapping.
- Handles long-range credit assignment.

Open questions.

- You must *know* a good target return.
- Limited extrapolation beyond the data.
- No optimality guarantees; heavy on context length.

Discuss. Where would you trust a return-conditioned model, and where would you still want an explicit value function you can reason about?