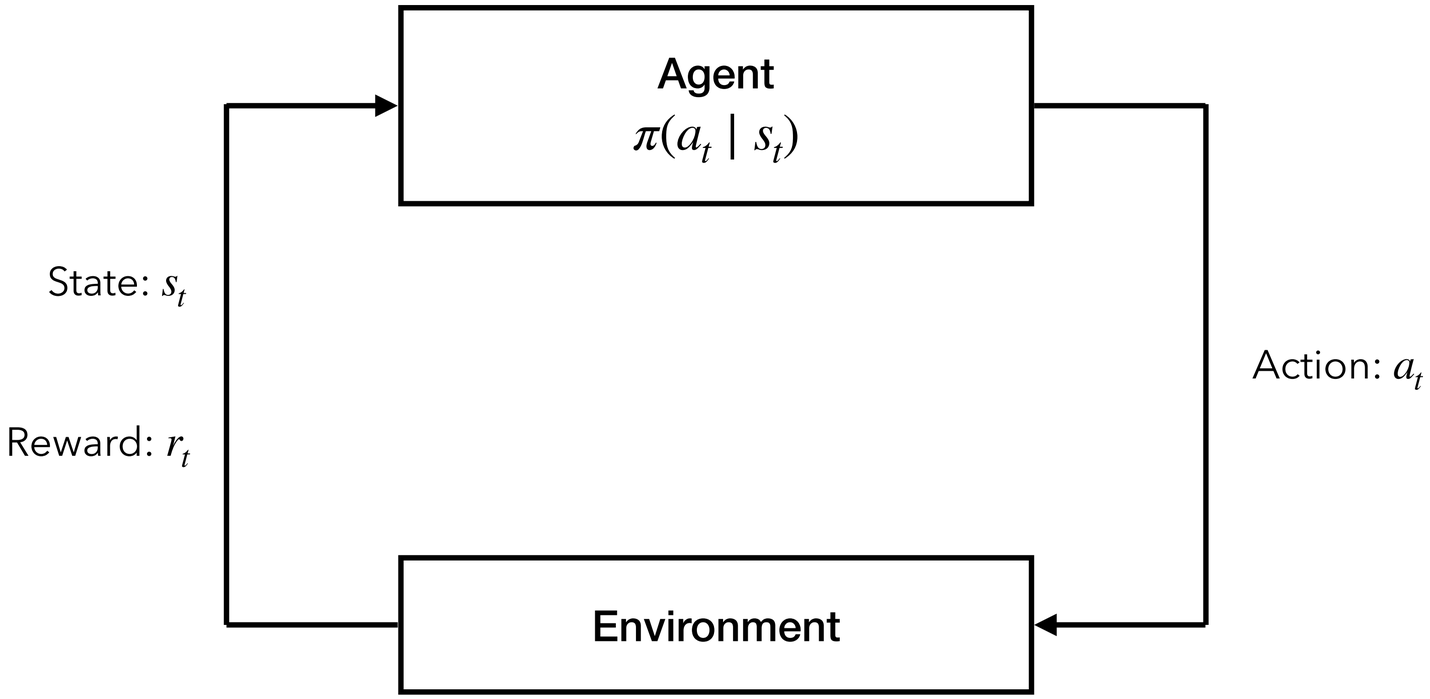
DS-GA 1003: Machine Learning

Lecture 12: Reinforcement Learning and Large Language Models

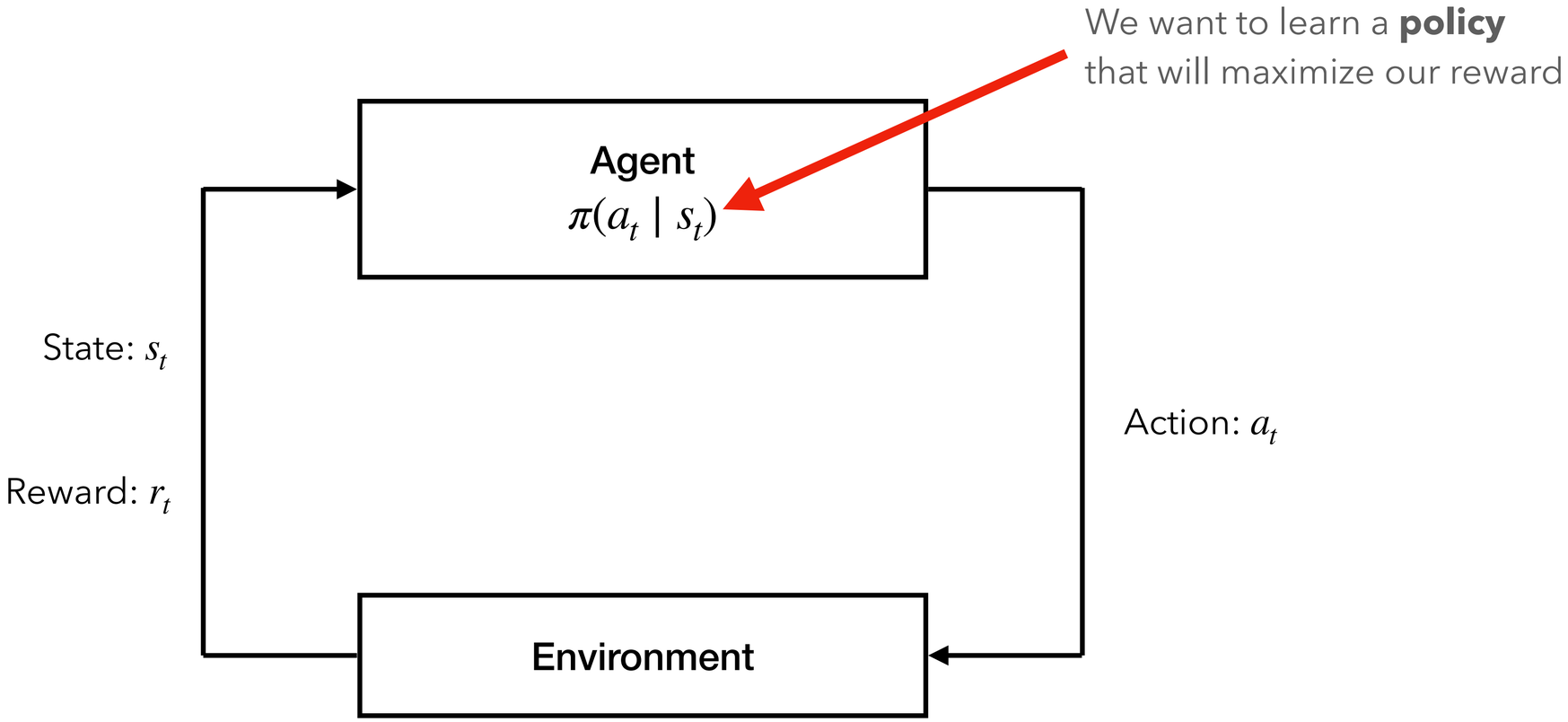
Slides adapted from: Sergey Levine, Dan Klein

Part I: Reinforcement Learning

The Reinforcement Learning Problem



The Reinforcement Learning Problem



The Reinforcement Learning Problem

State:

Action:

Reward:

The Reinforcement Learning Problem

State: <sensor values>

Action:

Reward:

The Reinforcement Learning Problem

State: <sensor values>

Action: [turn wheel 5° left, etc.]

Reward:

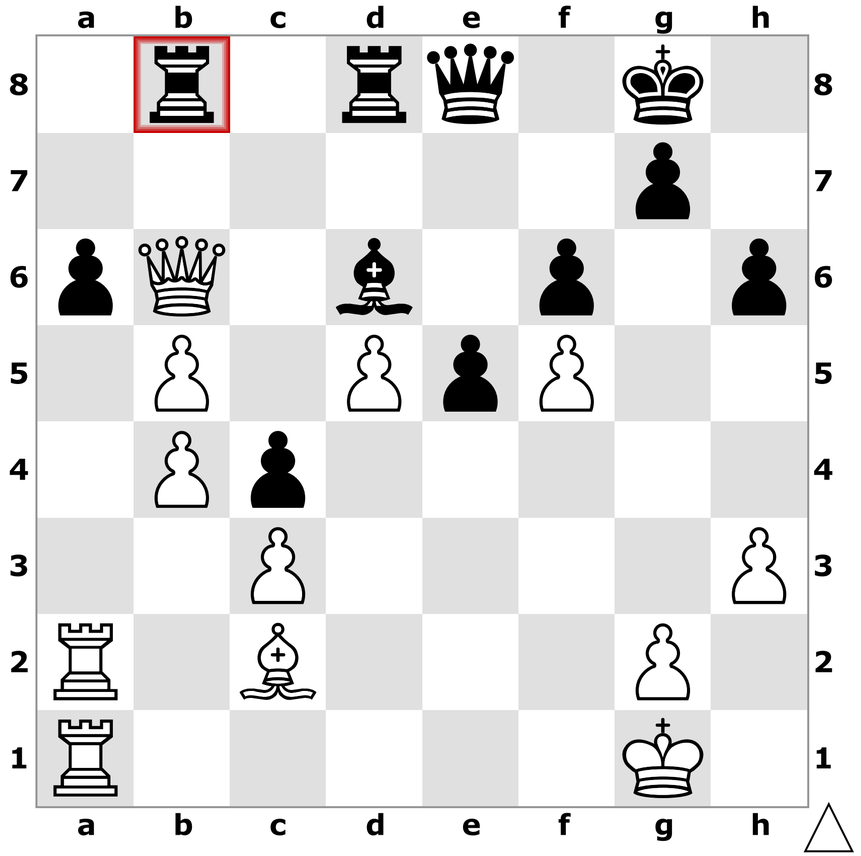
The Reinforcement Learning Problem

State: <sensor values>

Action: [turn wheel 5° left, etc.]

Reward: +1 if crash == False

The Reinforcement Learning Problem

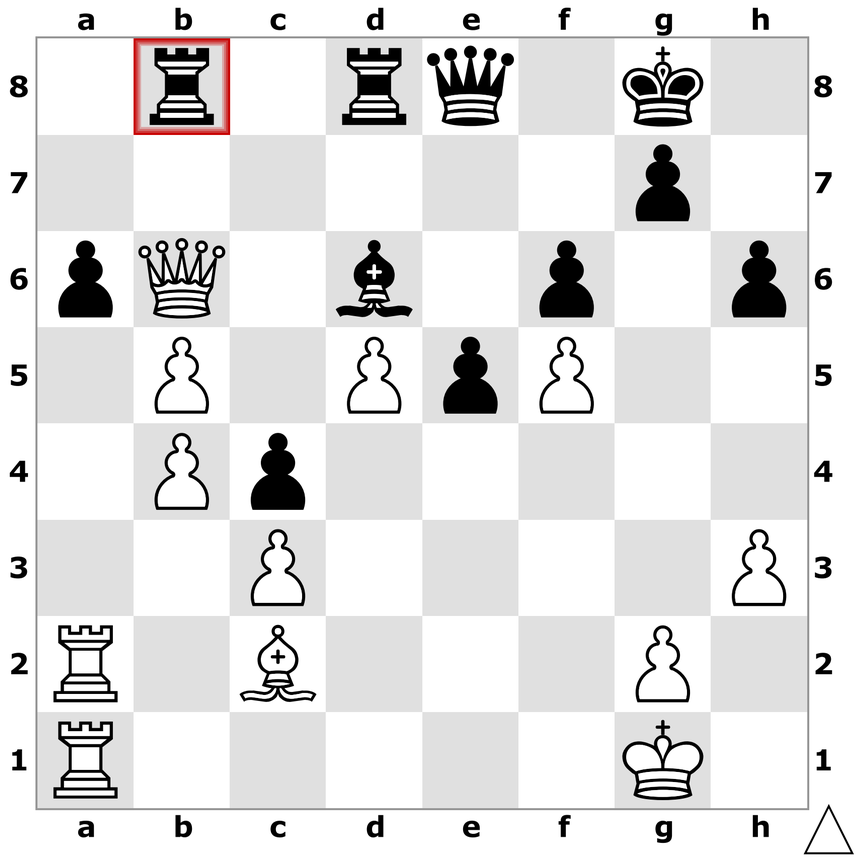


State:

Action:

Reward:

The Reinforcement Learning Problem

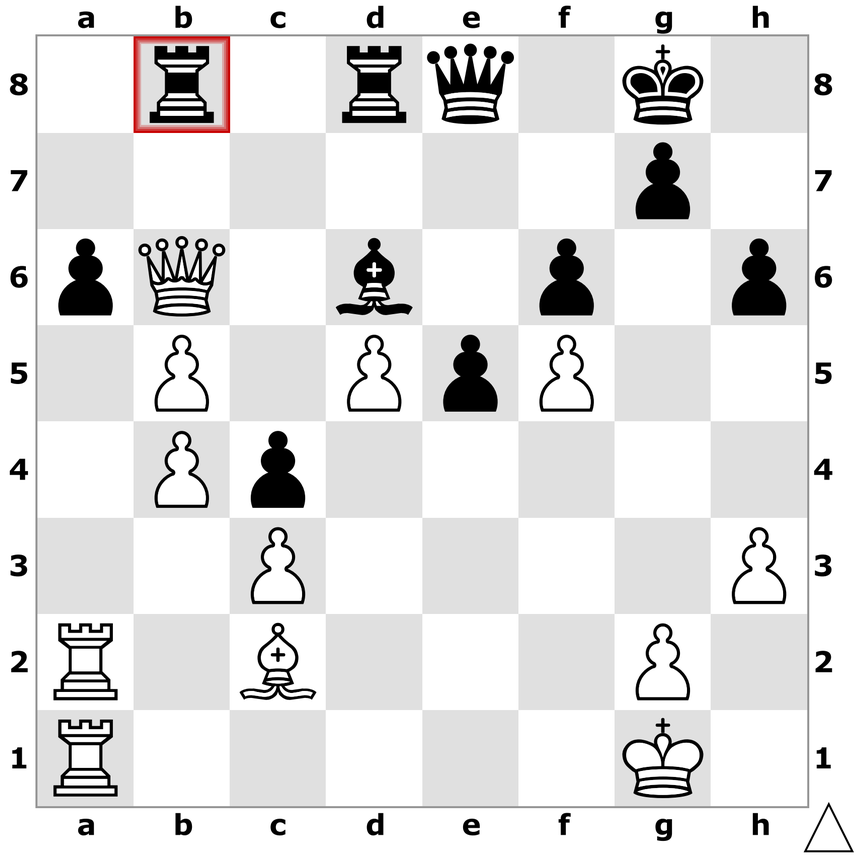


State: <positions of all pieces>

Action:

Reward:

The Reinforcement Learning Problem

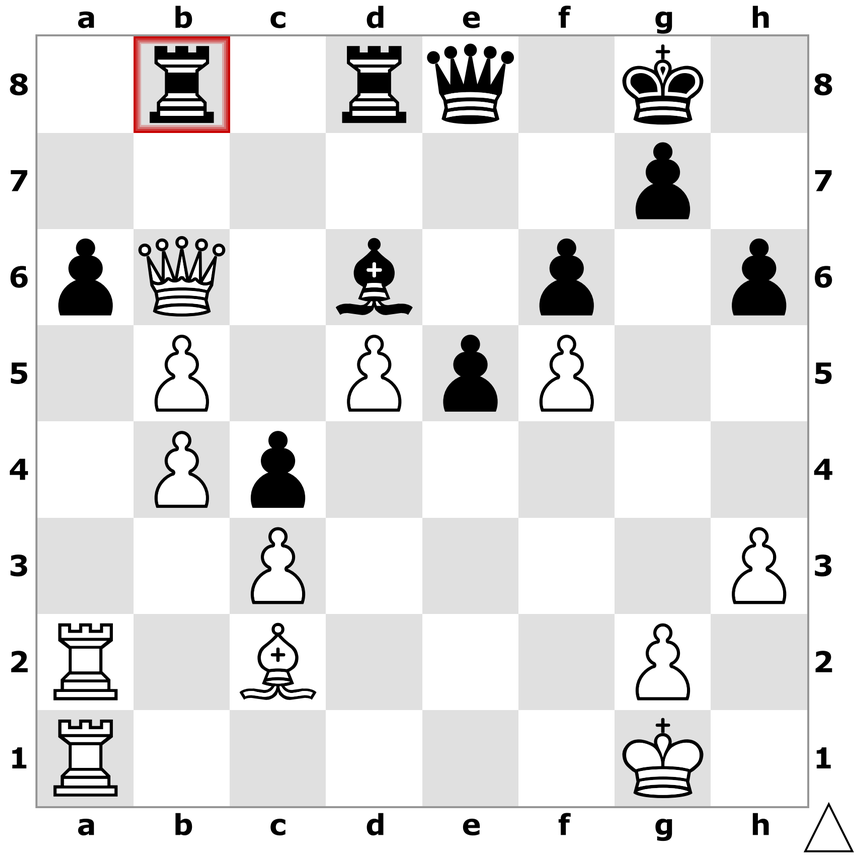


State: <positions of all pieces>

Action: [Qxa6, Qc6, etc.]

Reward:

The Reinforcement Learning Problem



State: <positions of all pieces>

Action: [Qxa6, Qc6, etc.]

Reward: +1 if checkmate, 0 if draw

Behavior Cloning

A very simple recipe:

- Collect a large number of trajectories (e.g., full chess games) from humans
- Convert each trajectory into a sequence of (state, action) pairs
- Train a supervised learning model to predict p(action | state)

Behavior Cloning

A very simple recipe:

- Collect a large number of trajectories (e.g., full chess games) from humans
- Convert each trajectory into a sequence of (state, action) pairs
- Train a supervised learning model to predict p(action | state)

Filtered BC: only use a subset of (state, action) pairs (e.g., those from winning games)

Behavior Cloning

Question: Does behavior cloning work?

Behavior Cloning

Question: Does behavior cloning work?

Answer: Depends on the domain

Behavior Cloning

Question: Does behavior cloning work?

Answer: Depends on the domain

Transcendence: Generative Models Can Outperform The Experts That Train Them

Edwin Zhang
OpenAI
Harvard University
Humanity Unleashed
edwin@openai.com

Vincent Zhu
UC Santa Barbara
Humanity Unleashed
vincentzhu@ucsb.edu

Naomi Saphra
Harvard University
Kempner Institute
nsaphra@g.harvard.edu

Anat Kleiman
Harvard University
Apple
anatkleiman@g.harvard.edu

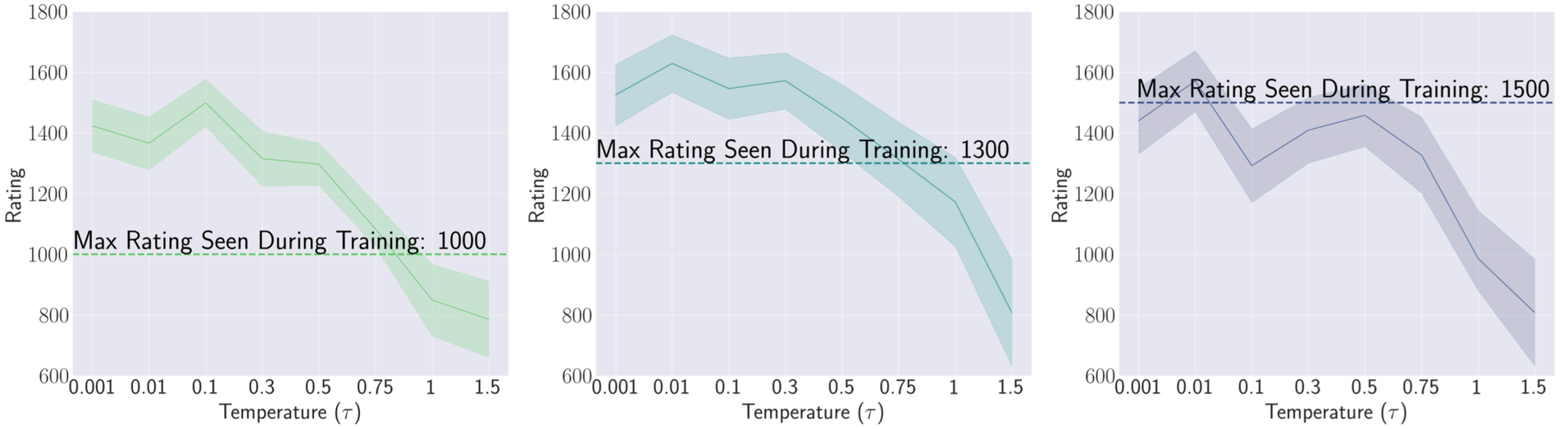
Benjamin L. Edelman
Princeton University
Harvard University
bedelman@g.harvard.edu

Milind Tambe
Harvard University
tambe@g.harvard.edu

Sham Kakade
Harvard University
Kempner Institute
sham@g.harvard.edu

Eran Malach
Harvard University
Kempner Institute
emalach@g.harvard.edu

BC works if you're training a chess agent on novice data!



Behavior Cloning

Question: Does behavior cloning work?

Answer: Depends on the domain

Trains on 15 billion data points!

- Stockfish 16: 2706 Elo
- BC model: 2299 Elo

Google DeepMind

Grandmaster-Level Chess Without Search

Anian Ruoss[*,1], **Grégoire Delétang**[*,1], **Sourabh Medapati**[1], **Jordi Grau-Moya**[1], **Li Kevin Wenliang**[1], **Elliot Catt**[1], **John Reid**[1] **and Tim Genewein**[1]

[*]Equal contributions, [1]Google DeepMind

The recent breakthrough successes in machine learning are mainly attributed to scale: namely large-scale attention-based architectures and datasets of unprecedented scale. This paper investigates the impact of training at scale for chess. Unlike traditional chess engines that rely on complex heuristics, explicit search, or a combination of both, we train a 270M parameter transformer model with supervised learning on a dataset of 10 million chess games. We annotate each board in the dataset with action-values provided by the powerful Stockfish 16 engine, leading to roughly 15 billion data points. Our largest model reaches a Lichess blitz Elo of 2895 against humans, and successfully solves a series of challenging chess puzzles, without any domain-specific tweaks or explicit search algorithms. We also show that our model outperforms AlphaZero's policy and value networks (without MCTS) and GPT-3.5-turbo-instruct. A systematic investigation of model and dataset size shows that strong chess performance only arises at sufficient scale. To validate our results, we perform an extensive series of ablations of design choices and hyperparameters.

5.1. Limitations

While our largest model achieves very good performance, it does not completely close the gap to Stockfish 16. All our scaling experiments point towards closing this gap eventually with a large enough model

Behavior Cloning

Question: Does behavior cloning work?

Answer: Depends on the domain

Trains on 15 billion data points!

- Stockfish 16: 2706 Elo
- BC model: 2299 Elo

We can’t always assume access to a stronger model!

Google DeepMind

Grandmaster-Level Chess Without Search

Anian Ruoss[*,1], **Grégoire Delétang**[*,1], **Sourabh Medapati**[1], **Jordi Grau-Moya**[1], **Li Kevin Wenliang**[1], **Elliot Catt**[1], **John Reid**[1] **and Tim Genewein**[1]
[*]Equal contributions, [1]Google DeepMind

The recent breakthrough successes in machine learning are mainly attributed to scale: namely large-scale attention-based architectures and datasets of unprecedented scale. This paper investigates the impact of training at scale for chess. Unlike traditional chess engines that rely on complex heuristics, explicit search, or a combination of both, we train a 270M parameter transformer model with supervised learning on a dataset of 10 million chess games. We annotate each board in the dataset with action-values provided by the powerful Stockfish 16 engine, leading to roughly 15 billion data points. Our largest model reaches a Lichess blitz Elo of 2895 against humans, and successfully solves a series of challenging chess puzzles, without any domain-specific tweaks or explicit search algorithms. We also show that our model outperforms AlphaZero’s policy and value networks (without MCTS) and GPT-3.5-turbo-instruct. A systematic investigation of model and dataset size shows that strong chess performance only arises at sufficient scale. To validate our results, we perform an extensive series of ablations of design choices and hyperparameters.

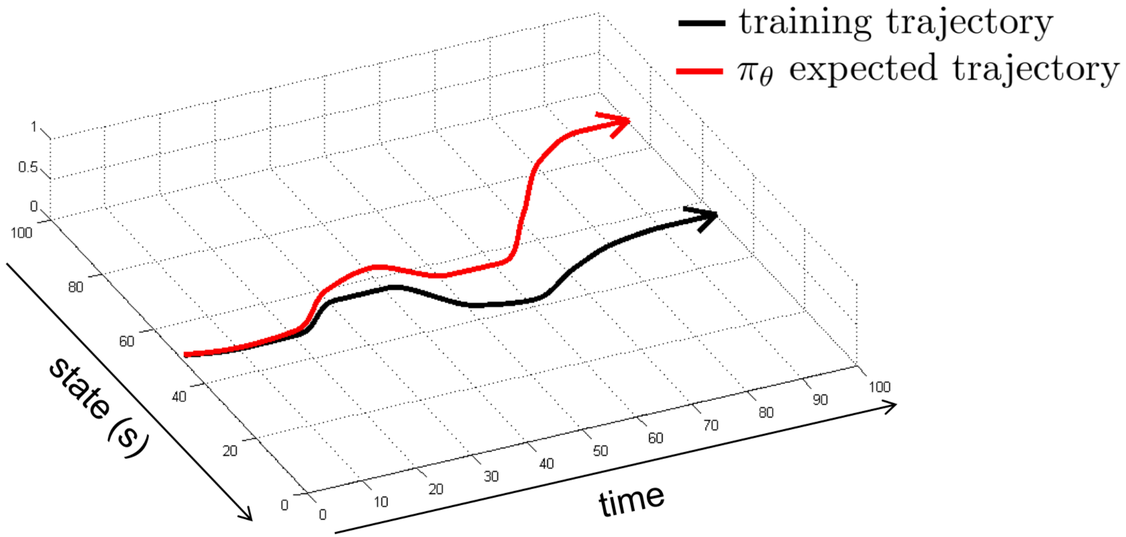
5.1. Limitations

While our largest model achieves very good performance, it does not completely close the gap to Stockfish 16. All our scaling experiments point towards closing this gap eventually with a large enough model

Behavior Cloning

Question: Does behavior cloning work?

Answer: Depends on the domain

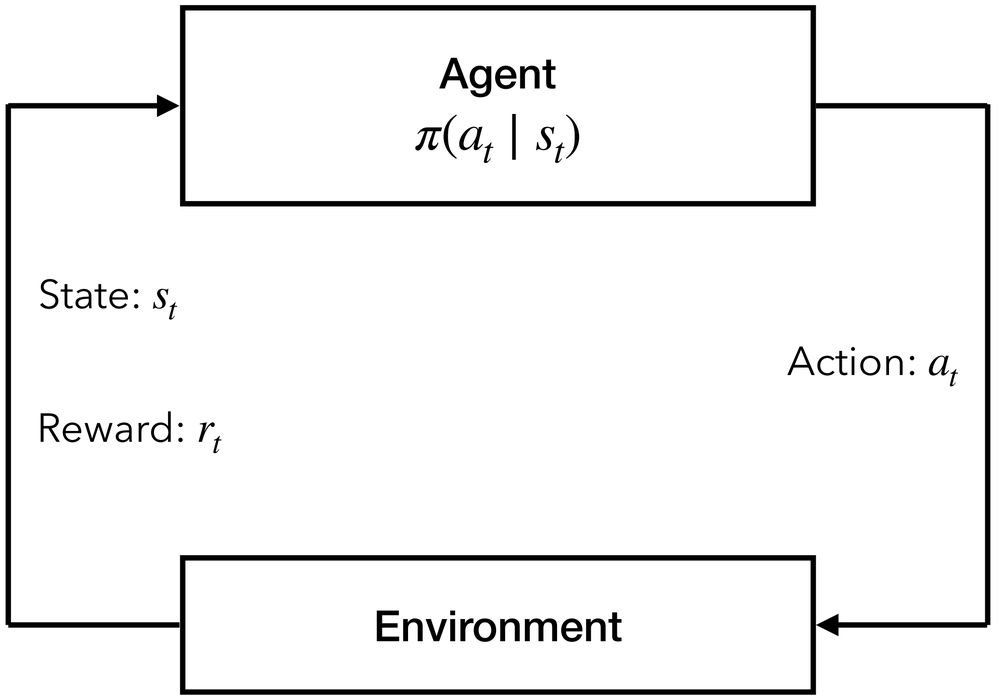


The Compounding Mistakes Problem:

A model trained via behavior cloning in a high-dimensional state space will gradually veer away from states which are supported in the training data, leading to larger mistakes

We Need a New Framework

Goal: we want to model sequential structure and explicitly optimize for rewards.

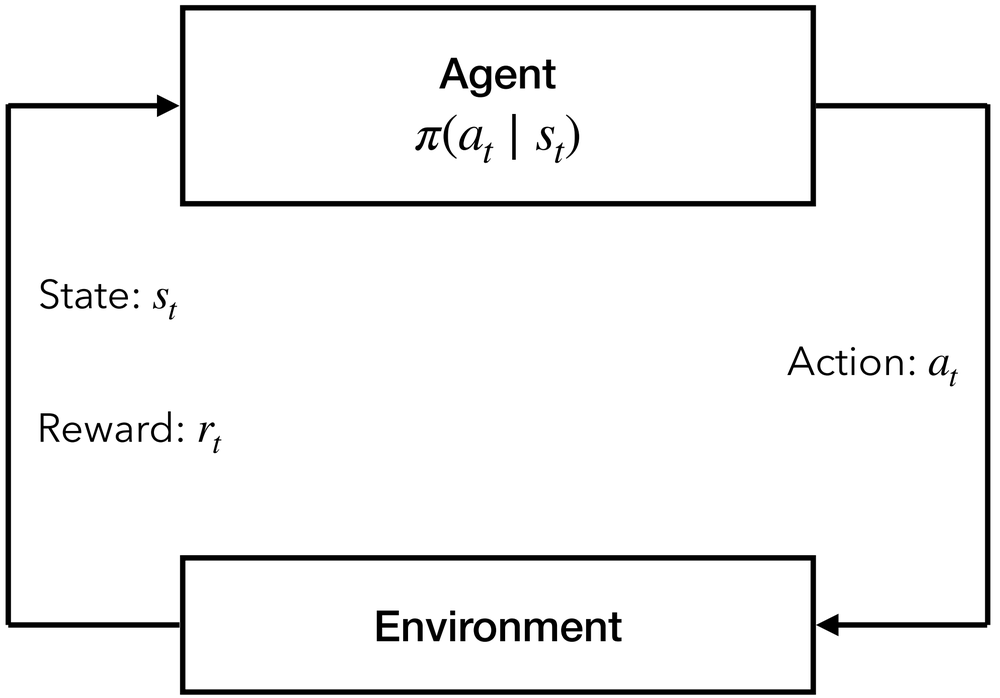


We Need a New Framework

Goal: we want to model sequential structure and explicitly optimize for rewards.

Markov Decision Process:

- State space: $\mathcal{S}$
- Action space: $\mathcal{A}$
- Transition function: $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
- Reward function: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

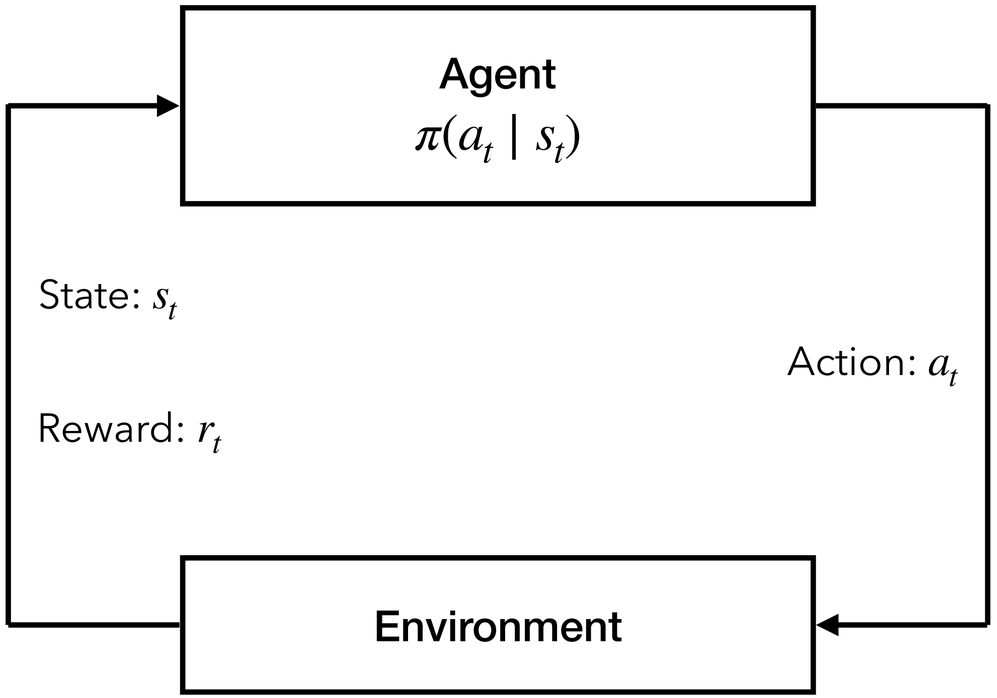


We Need a New Framework

Goal: we want to model sequential structure and explicitly optimize for rewards.

Markov Decision Process:

- State space: $\mathcal{S}$
- Action space: $\mathcal{A}$
- Transition function: $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
- Reward function: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



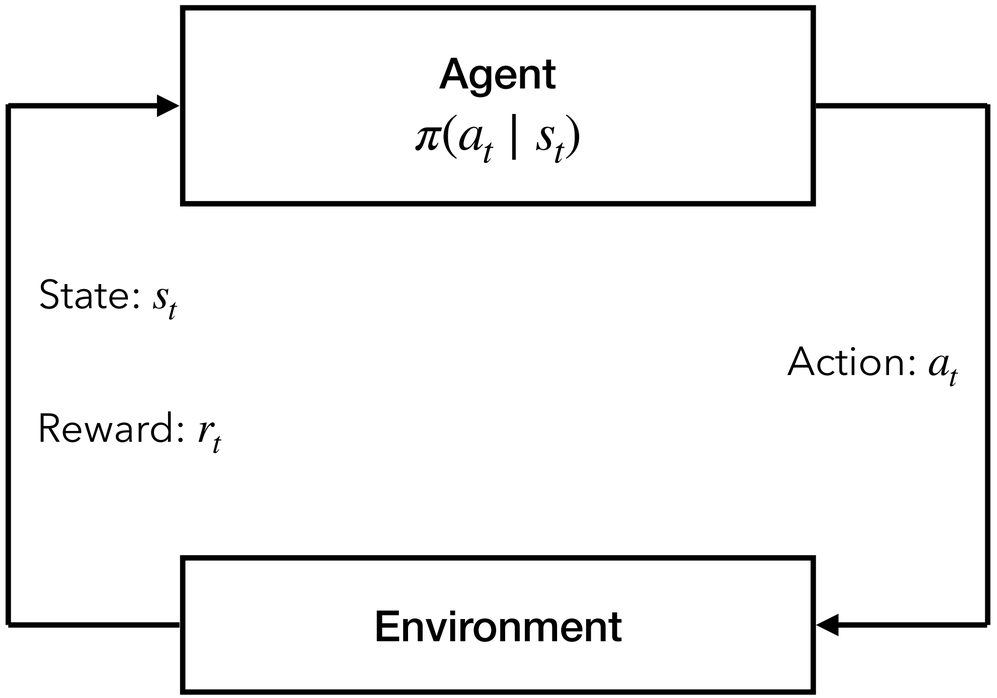
We want to optimize: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$

We Need a New Framework

Goal: we want to model sequential structure and explicitly optimize for rewards.

Markov Decision Process:

- State space: $\mathcal{S}$
- Action space: $\mathcal{A}$
- Transition function: $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
- Reward function: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$



We want to optimize: $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$ where γ is a discounting parameter that incentivizes nearby rewards

Policies and Values

Two different things we can optimize:

- Policy: $\pi(a \mid s)$ = a chosen action given a state

Policies and Values

Two different things we can optimize:

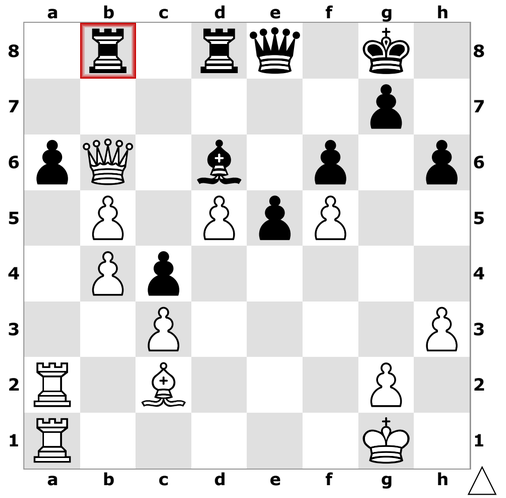
- Policy: $\pi(a \mid s)$ = a chosen action given a state
- Values:
  - $V(s)$ = expected utility of being in state s
  - $Q(s, a)$ = expected utility of being in state s and taking action a

Policies and Values

Two different things we can optimize:

- Policy: $\pi(a \mid s)$ = a chosen action given a state
- Values:
  - $V(s)$ = expected utility of being in state s
  - $Q(s, a)$ = expected utility of being in state s and taking action a

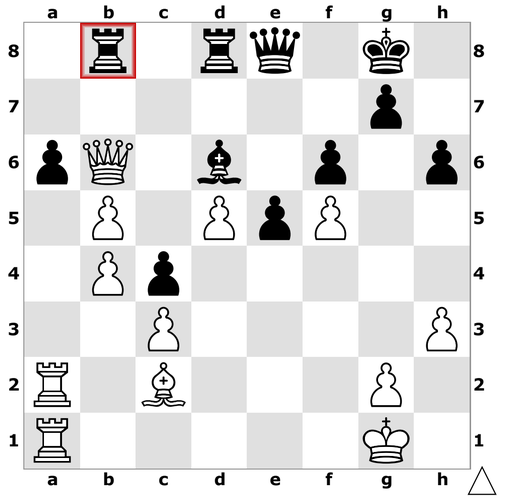
What's the V of this state?



What's Q(s, Qb7)?

Policies and Values

What's the V of this state?



What's Q(s, Qb7)?

Two different things we can optimize:

- Policy: $\pi(a \mid s)$ = a chosen action given a state
- Values:
  - $V(s)$ = expected utility of being in state s
  - $Q(s, a)$ = expected utility of being in state s and taking action a

Optimal Policies and Values

We want to learn:

- Policy: $\pi^*(a \mid s)$ = optimal action given a state
- Values:
  - $V^*(s)$ = expected utility of being in state s and taking optimal actions
  - $Q^*(s, a)$ = expected utility of being in state s, taking action a, and (thereafter) acting optimally

Optimal Policies and Values

We want to learn:

- Policy: $\pi^*(a \mid s)$ = optimal action given a state
- Values:
  - $V^*(s)$ = expected utility of being in state s and taking optimal actions
  - $Q^*(s, a)$ = expected utility of being in state s, taking action a, and (thereafter) acting optimally

These representations are somewhat interchangeable!

Optimal Policies and Values

Given an optimal value function Q^*, we can derive the optimal policy:

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Optimal Policies and Values

Given an optimal value function Q^*, we can derive the optimal policy:

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

More complicated, but we can also derive a policy from V^*:

Optimal Policies and Values

Given an optimal value function Q^*, we can derive the optimal policy:

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

More complicated, but we can also derive a policy from V^*:

$$\pi(s) = \arg\max_a \left[\sum_{s' \in \mathcal{S}} T(s' \mid s, a) \cdot V^*(s') \right]$$

The Bellman Equations

Sometimes it's easier to learn value functions, because we can define them recursively:

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s'} T(s' \mid s, a) \left[R(s, a, s') + \gamma V^{\pi}(s') \right]$$

$$Q^{\pi}(s, a) = \sum_{s'} T(s' \mid s, a) \left[R(s, a, s') + \gamma \sum_{a'} \pi(a' \mid s') Q^{\pi}(s', a') \right]$$

The Bellman Equations

…and we can define optimality equations similarly:

$$V^*(s) = \max_a \sum_{s'} T(s' \mid s, a)\left[R(s, a, s') + \gamma V^*(s')\right]$$

$$Q^*(s, a) = \sum_{s'} T(s' \mid s, a)\left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a')\right]$$

Value Iteration

A simple algorithm for learning value functions:

Value Iteration

A simple algorithm for learning value functions:

1. Start with $V_0(s) = 0$ for all states

Value Iteration

A simple algorithm for learning value functions:

1. Start with $V_0(s) = 0$ for all states

2. Given previous values, update them based on expected reward:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s' \mid s, a) \cdot \left[R(s, a) + \gamma V_k(s') \right]$$

Value Iteration

A simple algorithm for learning value functions:

1. Start with $V_0(s) = 0$ for all states
2. Given previous values, update them based on expected reward:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s' \mid s, a) \cdot \left[R(s, a) + \gamma V_k(s') \right]$$

3. Repeat until convergence

Value Iteration

A simple algorithm for learning value functions:

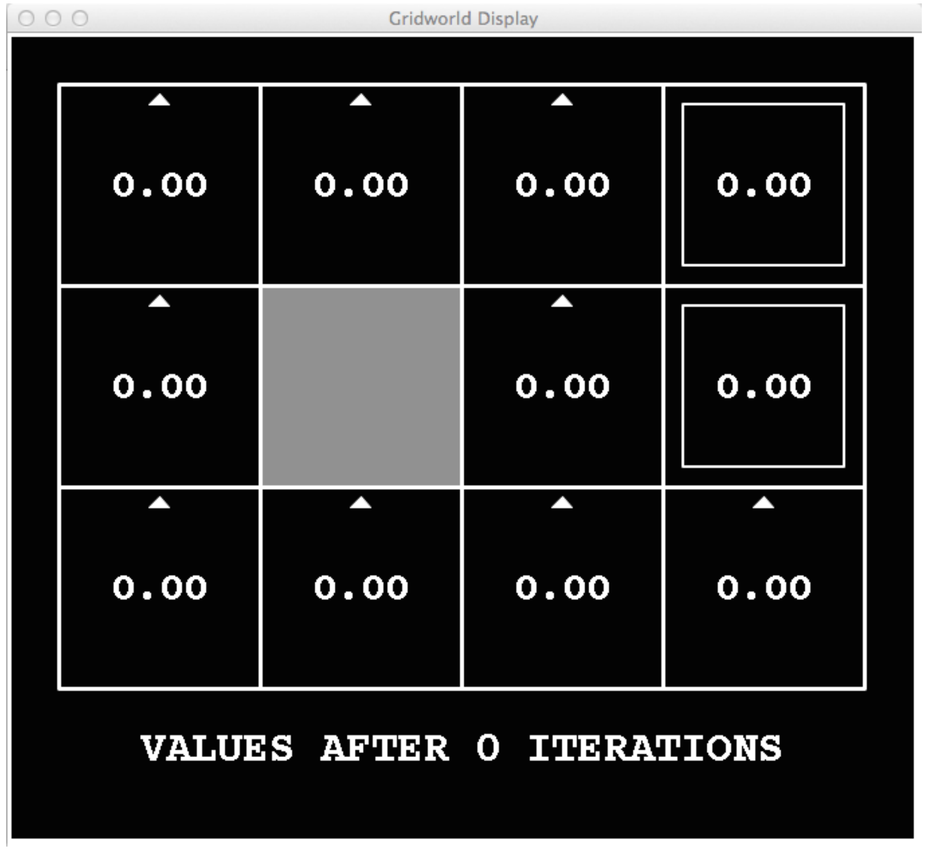
1. Start with $V_0(s) = 0$ for all states
2. Given previous values, update them based on expected reward:

$$V_{k+1}(s) = \max_a \sum_{s'} T(s' \mid s, a) \cdot \left[R(s, a) + \gamma V_k(s') \right]$$

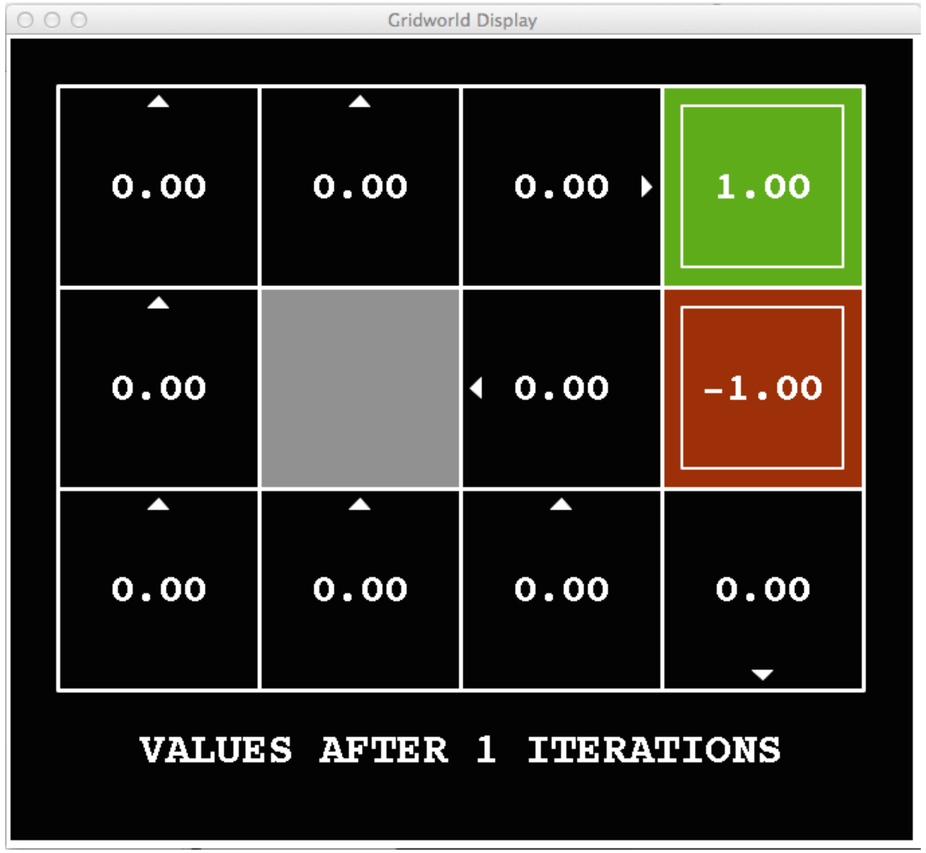
3. Repeat until convergence

Complexity of each iteration: $\mathcal{O}(|\mathcal{S}|^2 \cdot |\mathcal{A}|)$. But policy may converge before values do!

Value Iteration in Action



Value Iteration in Action

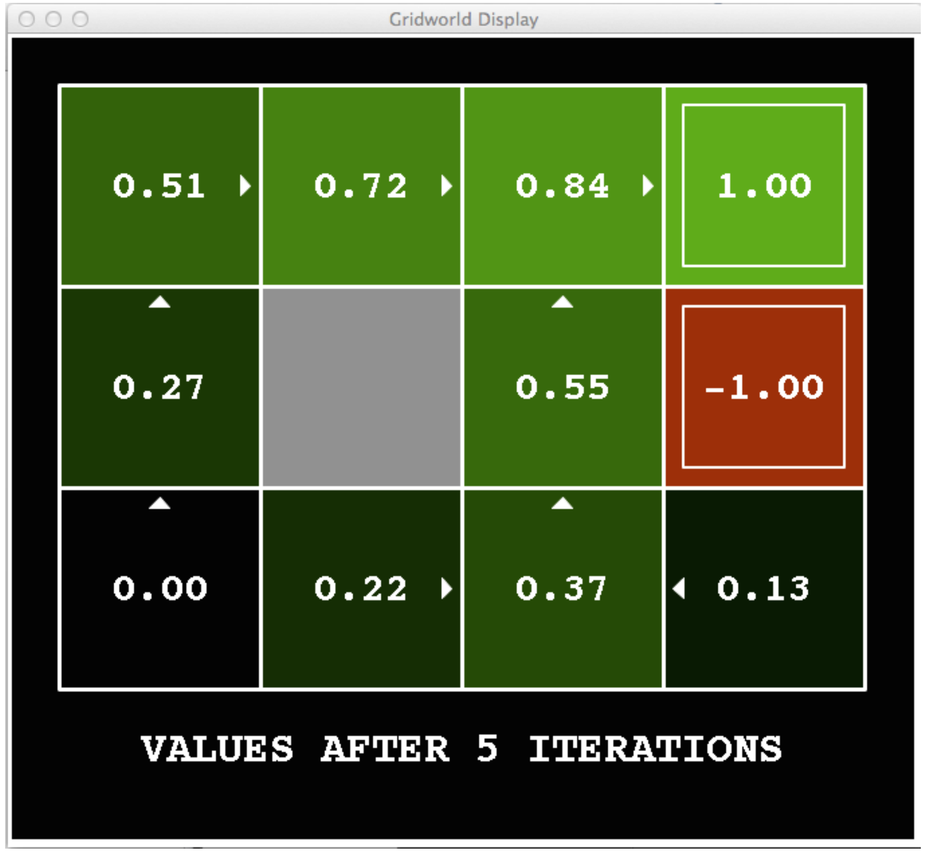


Value Iteration in Action

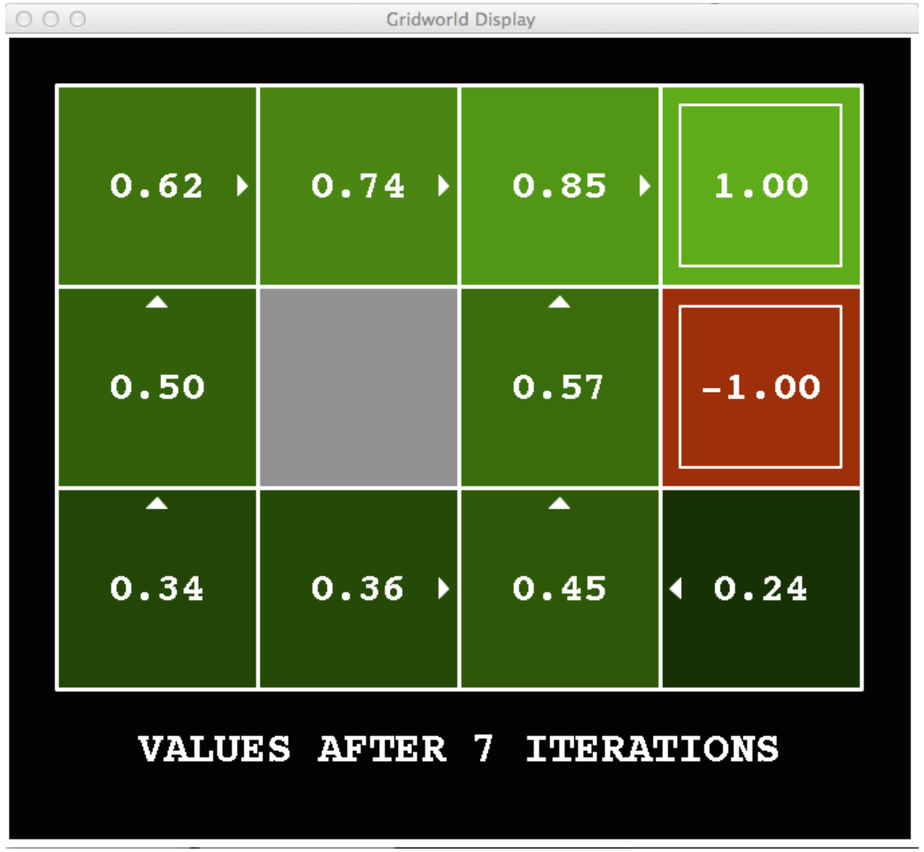
Value Iteration in Action

Value Iteration in Action

Value Iteration in Action



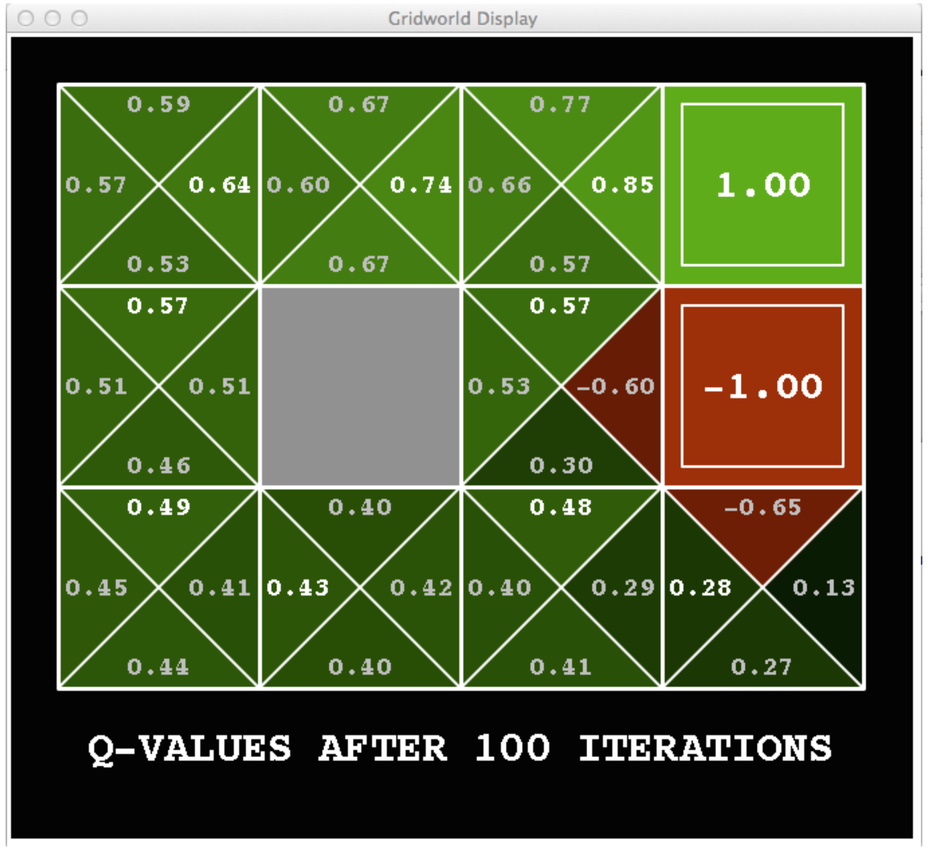
Value Iteration in Action



Value Iteration in Action

Value Iteration in Action

Value Iteration in Action



From Value Iteration to AlphaZero

Value iteration is a tabular method: need to store values for all (state, action) pairs

From Value Iteration to AlphaZero

Value iteration is a tabular method: need to store values for all (state, action) pairs

- This isn’t possible for a domain like chess or Go! The state space is too large

From Value Iteration to AlphaZero

Value iteration is a tabular method: need to store values for all (state, action) pairs

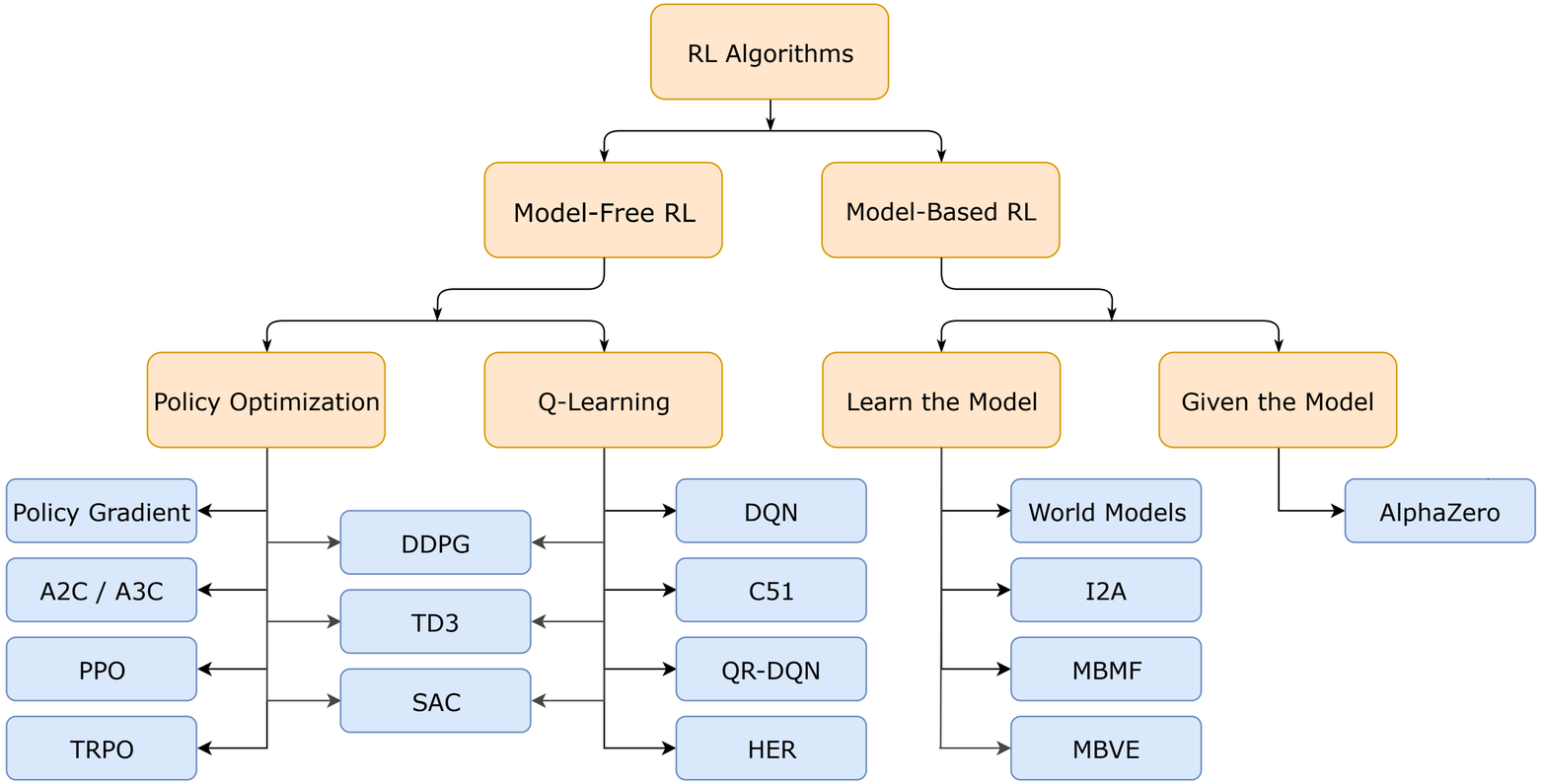
- This isn't possible for a domain like chess or Go! The state space is too large
- Instead: want to learn a neural network $V_\theta(s) \leftarrow \max_a \sum_{s'} T(s' \mid s, a) \cdot \left[R(s, a) + \gamma V_\theta(s')\right]$

From Value Iteration to AlphaZero

Value iteration is a tabular method: need to store values for all (state, action) pairs

- This isn't possible for a domain like chess or Go! The state space is too large
- Instead: want to learn a neural network $V_\theta(s) \leftarrow \max_a \sum_{s'} T(s' \mid s, a) \cdot \left[R(s, a) + \gamma V_\theta(s')\right]$
- In practice, we'll often learn $Q_\theta(s, a)$ instead:
  - Sample trajectories from the model and store them in a replay buffer
  - Instead of greedily sampling, do tree search to leverage more information in θ

Want to learn more about RL?



OpenAI: Spinning Up in Deep RL: https://spinningup.openai.com/en/latest/index.html

Deep Reinforcement Learning Course: https://rail.eecs.berkeley.edu/deeprlcourse/

Part II: LLMs

LLMs

- What are LLMs?
  - Closed-weight models: GPT-5, Claude, Gemini, Grok, …
  - Open-weight models: Qwen, Llama, GLM, OLMo, …

LLMs

- What are LLMs?
  - Closed-weight models: GPT-5, Claude, Gemini, Grok, …
  - Open-weight models: Qwen, Llama, GLM, OLMo, …

- Historically:
  - Input: a string of text
  - Output: a word (or a sequence of words)

LLMs

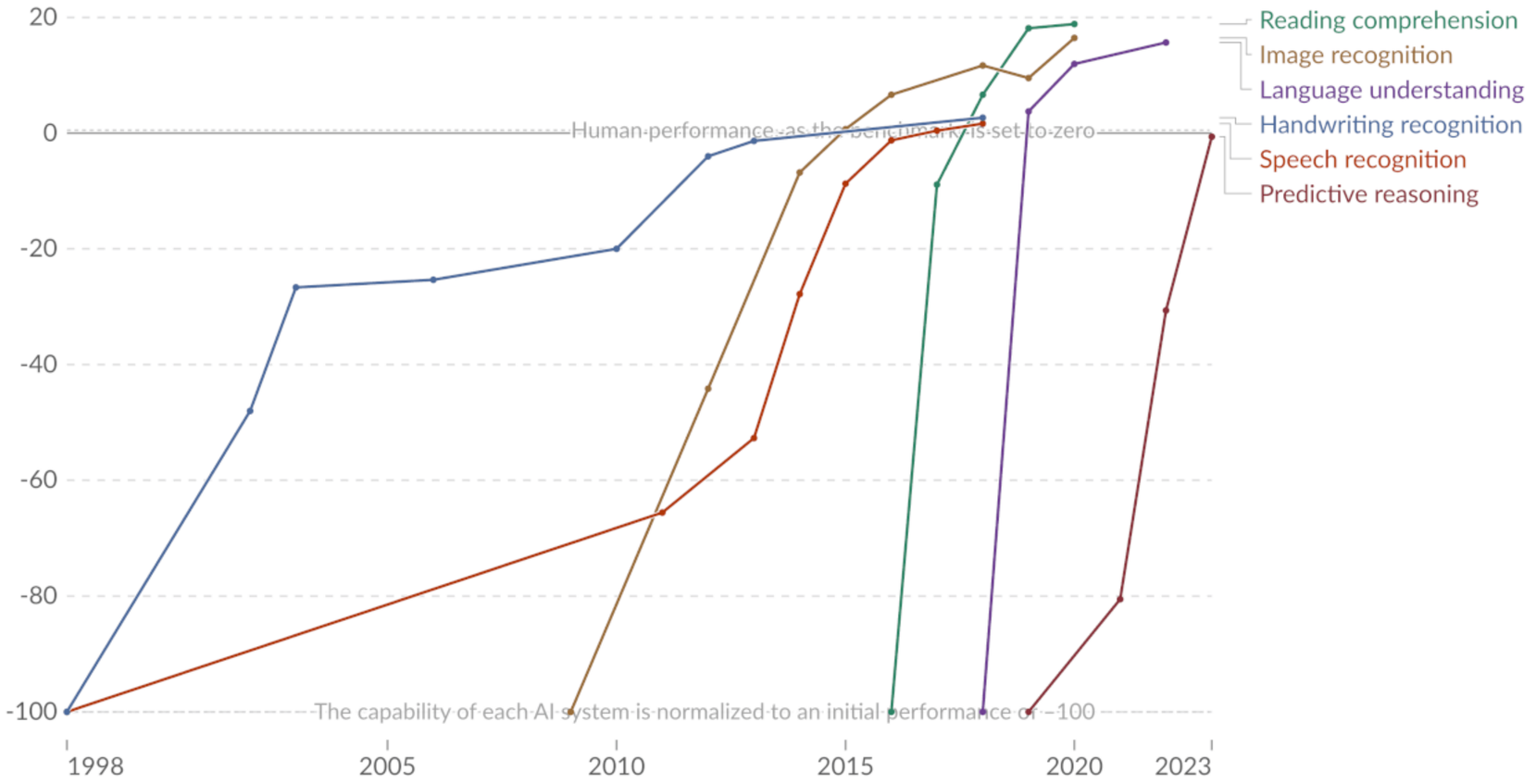
- What are LLMs?
  - Closed-weight models: GPT-5, Claude, Gemini, Grok, …
  - Open-weight models: Qwen, Llama, GLM, OLMo, …

- Historically:
  - Input: a string of text
  - Output: a word (or a sequence of words)

- Now: multimodal models (“foundation models”) with image/video inputs and outputs

Test scores of AI systems on various capabilities relative to human performance

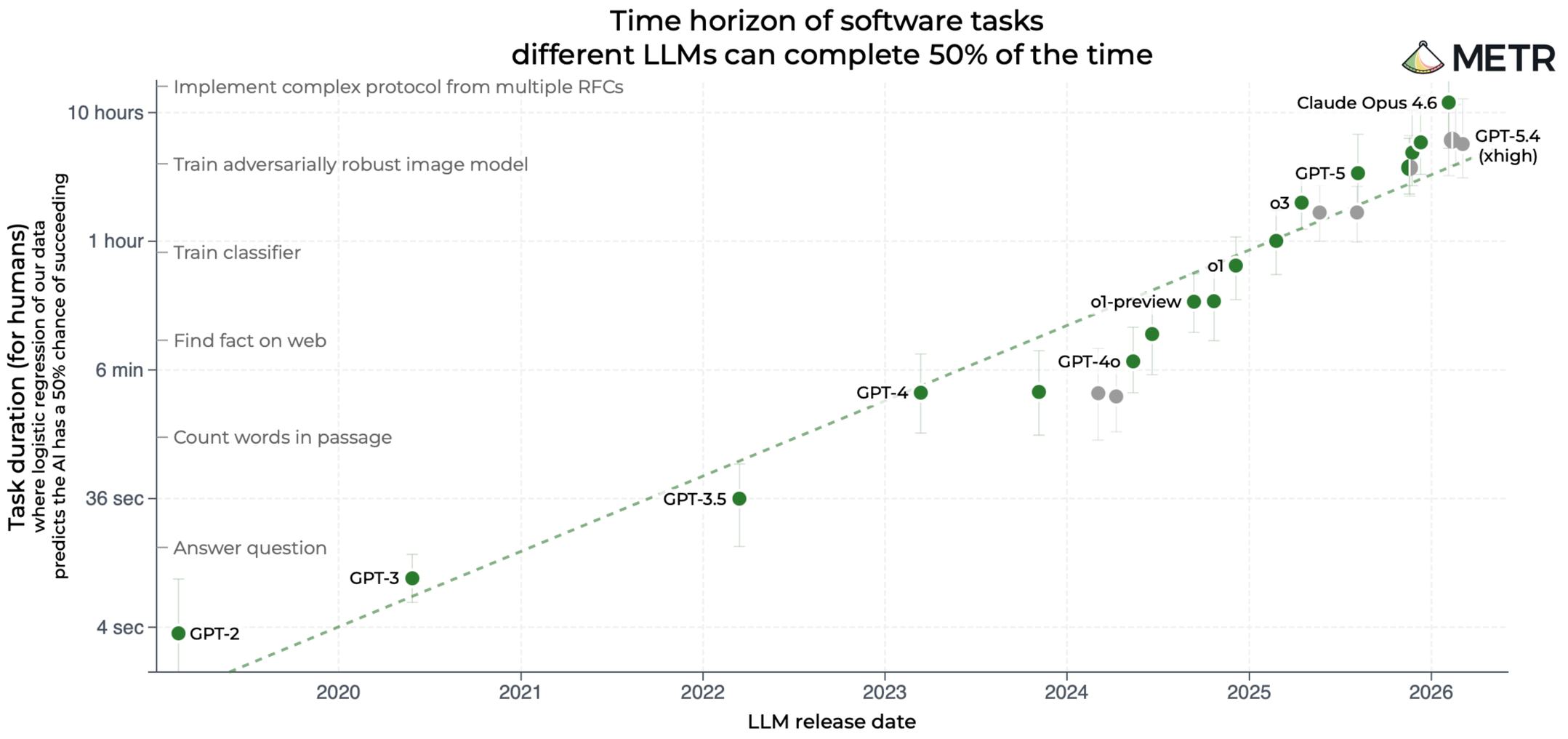
Within each domain, the initial performance of the AI is set to −100. Human performance is used as a baseline, set to zero. When the AI's performance crosses the zero line, it scored more points than humans.



Data source: Kiela et al. (2023)

OurWorldinData.org/artificial-intelligence | CC BY

Note: For each capability, the first year always shows a baseline of −100, even if better performance was recorded later that year.

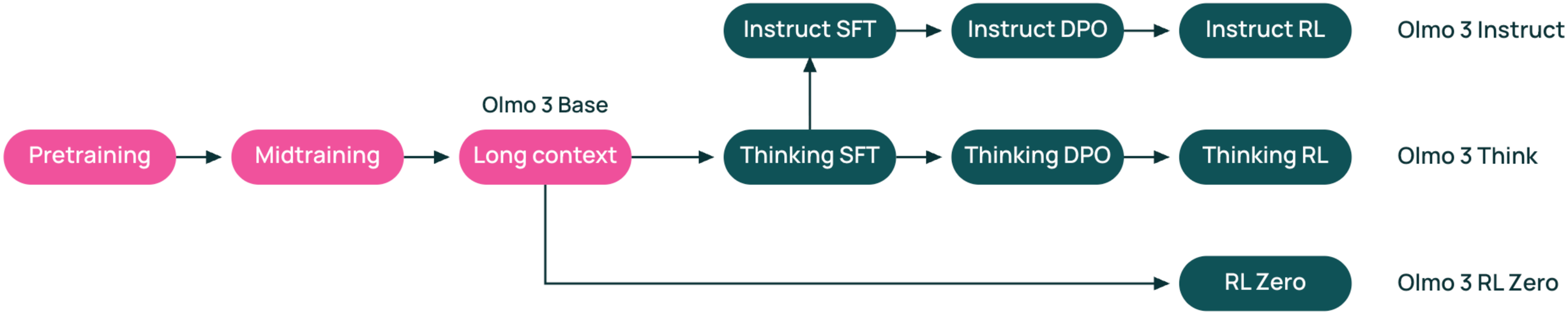

Time horizon of software tasks
different LLMs can complete 50% of the time
METR
Task duration (for humans)
where logistic regression of our data
predicts the AI has a 50% chance of succeeding
Implement complex protocol from multiple RFCs
10 hours
Train adversarially robust image model
1 hour
Train classifier
Find fact on web
6 min
Count words in passage
36 sec
Answer question
4 sec
GPT-2
GPT-3
GPT-3.5
GPT-4
GPT-4o
o1-preview
o1
o3
GPT-5
Claude Opus 4.6
GPT-5.4
(xhigh)
2020
2021
2022
2023
2024
2025
2026
LLM release date

What objective are we optimizing?

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

LLMs are trained in many different stages:

We don’t know what the exact stages are for closed-source models, but generally the pipeline looks like this:



Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

Next Token Prediction

The general recipe:

- Collect a very large set of documents for training (often terabytes)

Next Token Prediction

The general recipe:

- Collect a very large set of documents for training (often terabytes)
- Convert this into a supervised learning problem:
  - Input: a prefix of text
  - Output: the next word

Next Token Prediction

The general recipe:

- Collect a very large set of documents for training (often terabytes)
- Convert this into a supervised learning problem:
  - Input: a prefix of text “Behavior ” “Behavior cloning fails due to the compounding ”
  - Output: the next word “cloning” “error”

Next Token Prediction

The general recipe:

- Collect a very large set of documents for training (often terabytes)
- Convert this into a supervised learning problem:
  - Input: a prefix of text
  - Output: the next word
- Train a very large model on this supervised learning problem

N-Gram Language Models

Just use statistics from data to learn a table of word probabilities:

Bigram Model

198015222 the first
194623024 the same
168504105 the following
158562063 the world
…
14112454 the door

23135851162 the *

Trigram Model

197302 close the window
191125 close the door
152500 close the gap
116451 close the thread
87298 close the deal

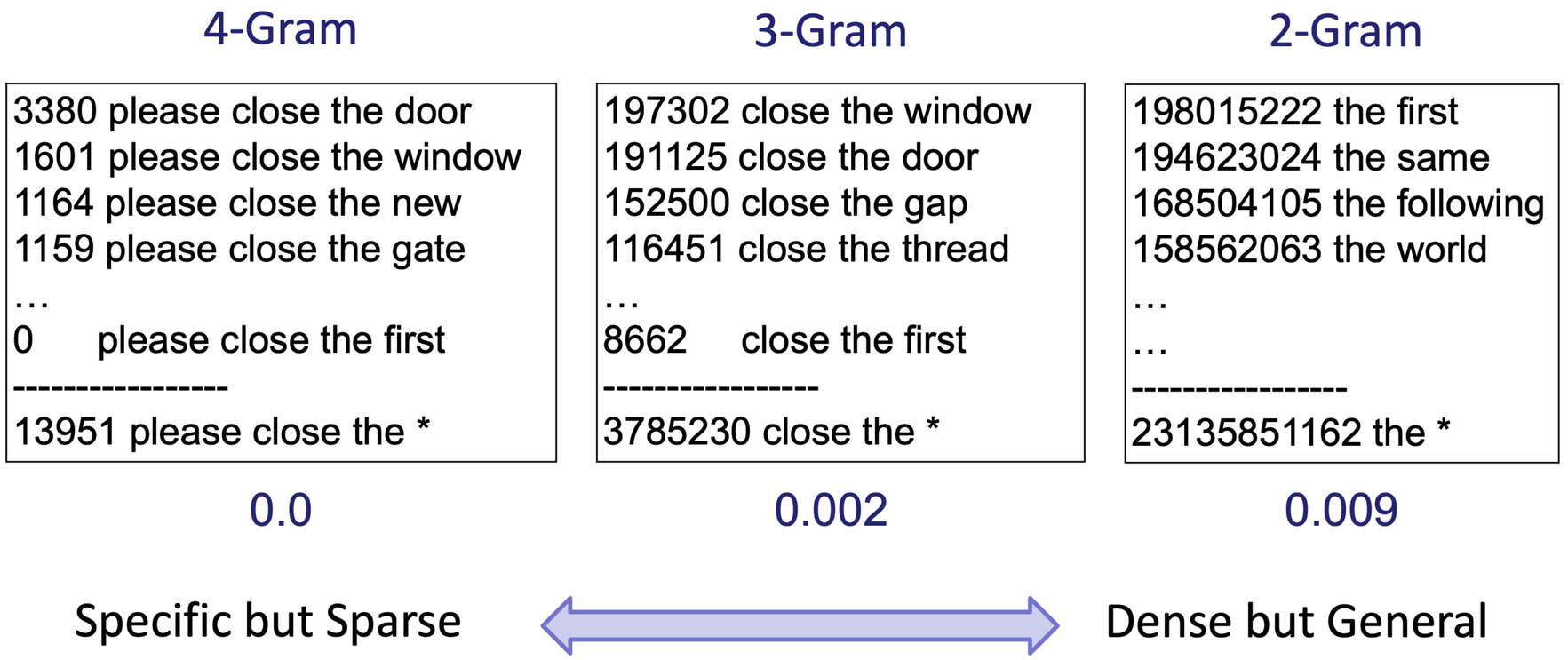
3785230 close the *

P(door | the) = 0.0006

P(door | close the) = 0.05

N-Gram Language Models with Backoff

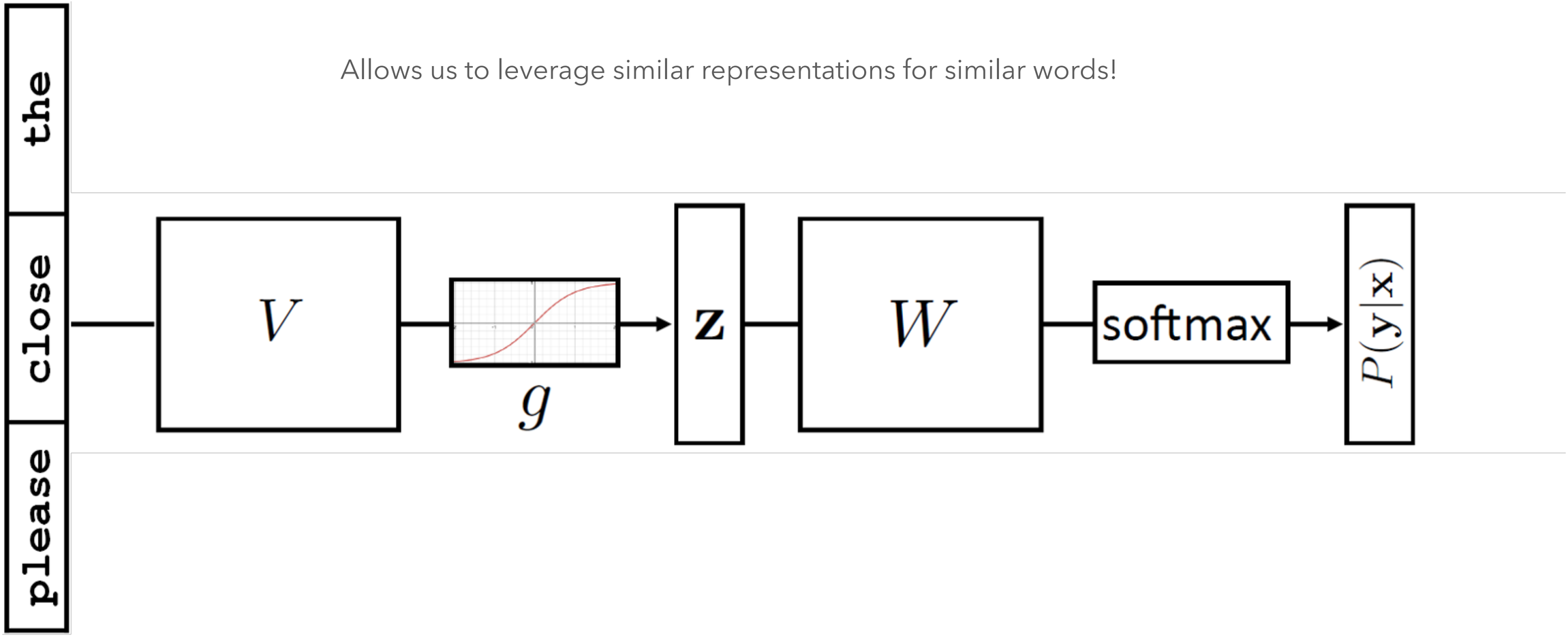
Interpolate between models with more context and less context:



$$\lambda \hat{P}(w|w_{-1}, w_{-2}) + \lambda' \hat{P}(w|w_{-1}) + \lambda'' \hat{P}(w)$$

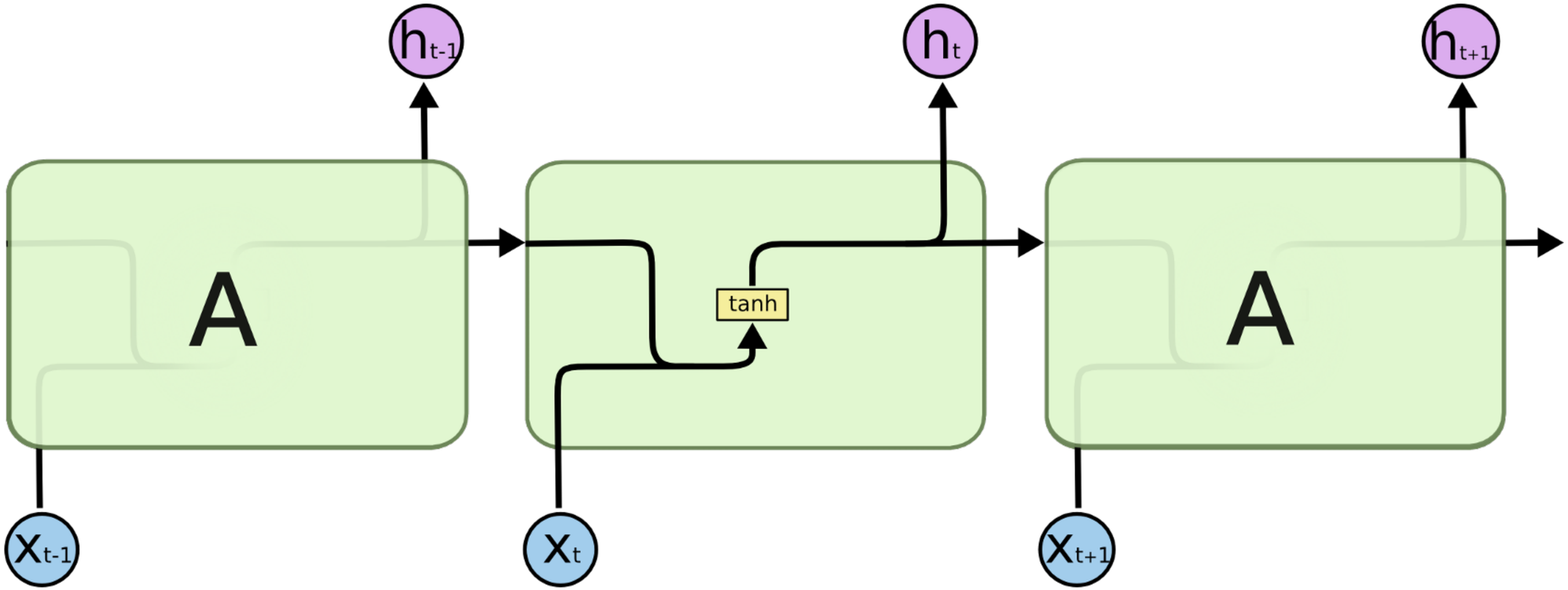
Recall: Neural N-Gram Models

Allows us to leverage similar representations for similar words!



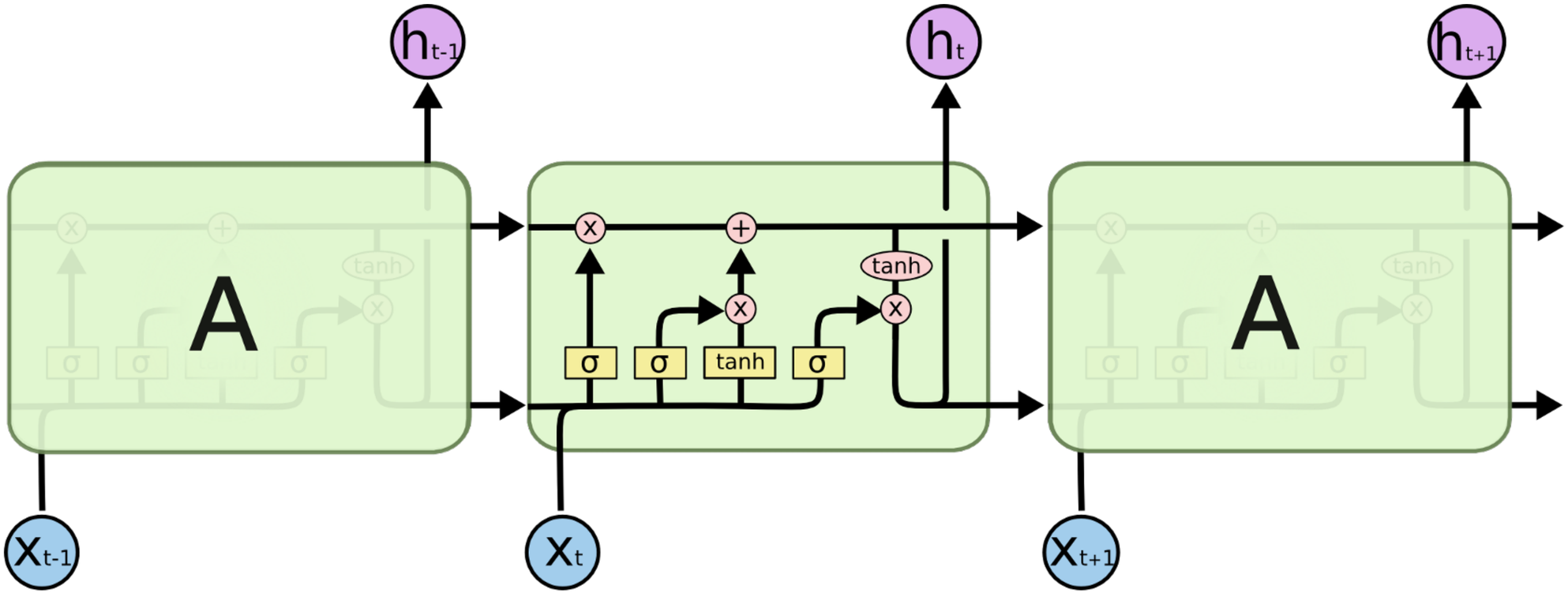
Recall: Recurrent Neural Networks

Allows us to handle variable length input sequences!

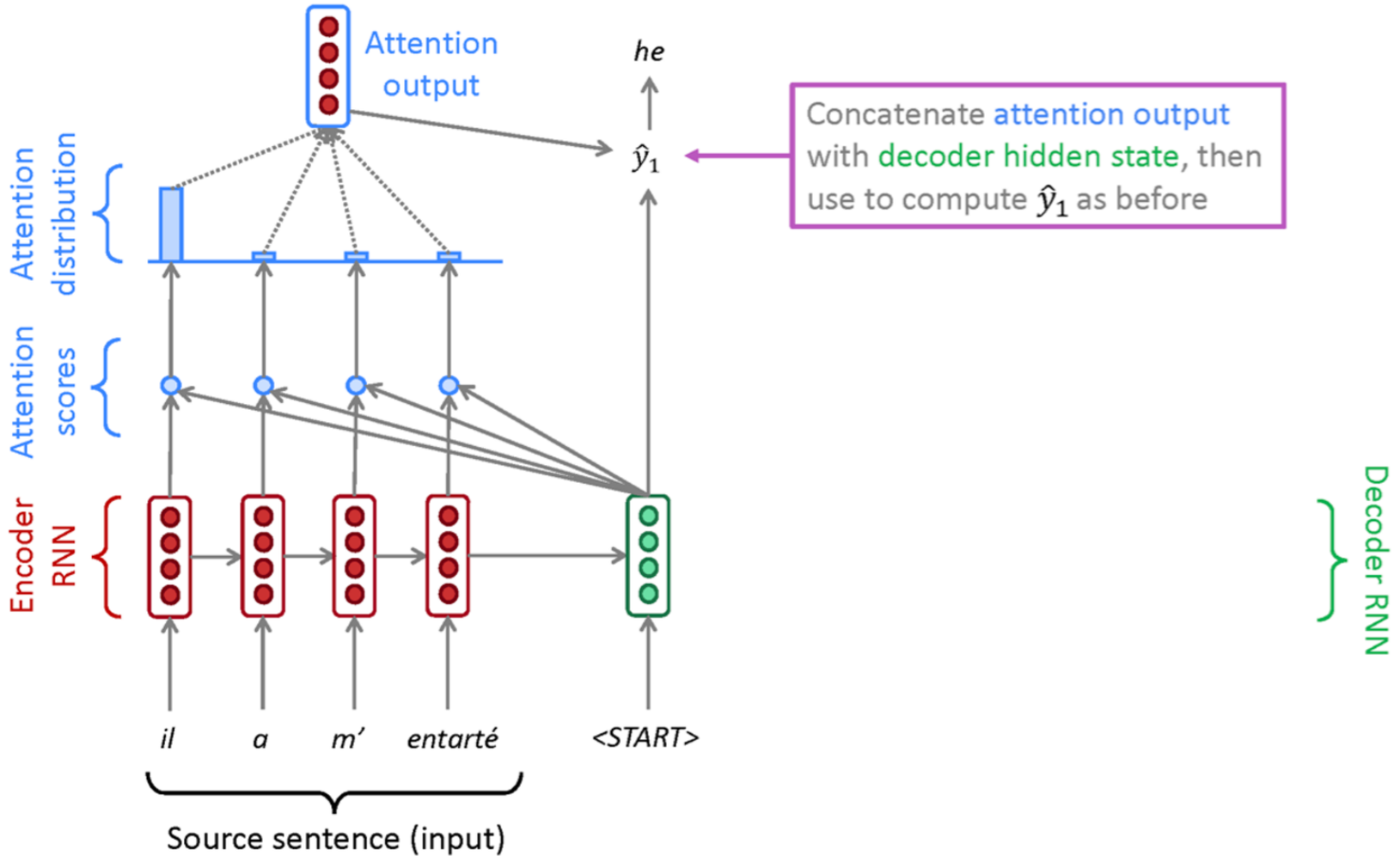


Recall: LSTMs

Allows us to handle variable length input sequences **more effectively**!



Recall: LSTMs with Attention

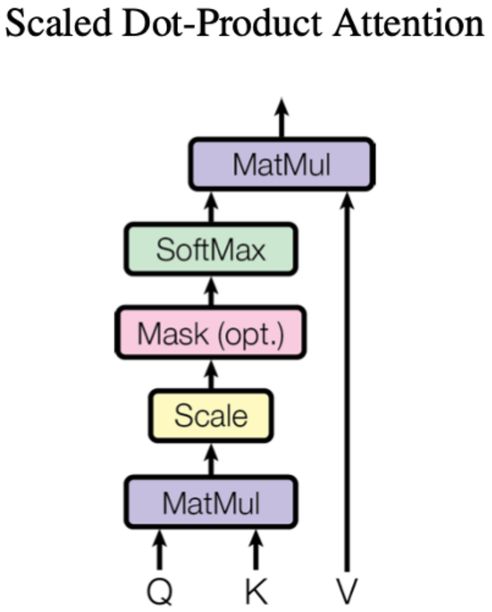


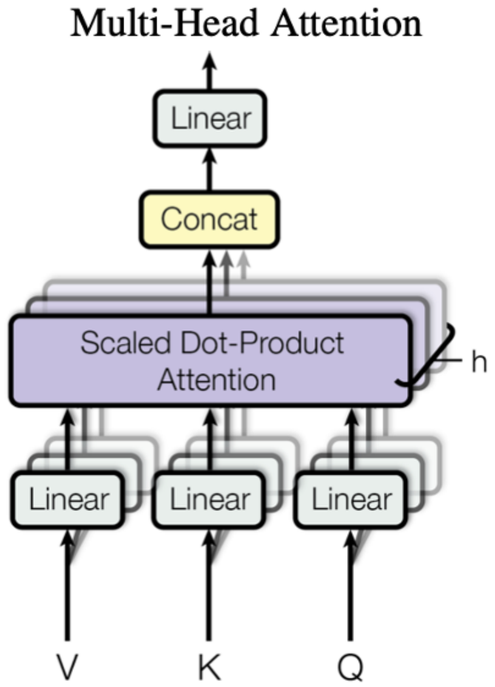
Allows us to condition next-word prediction on specific words from context

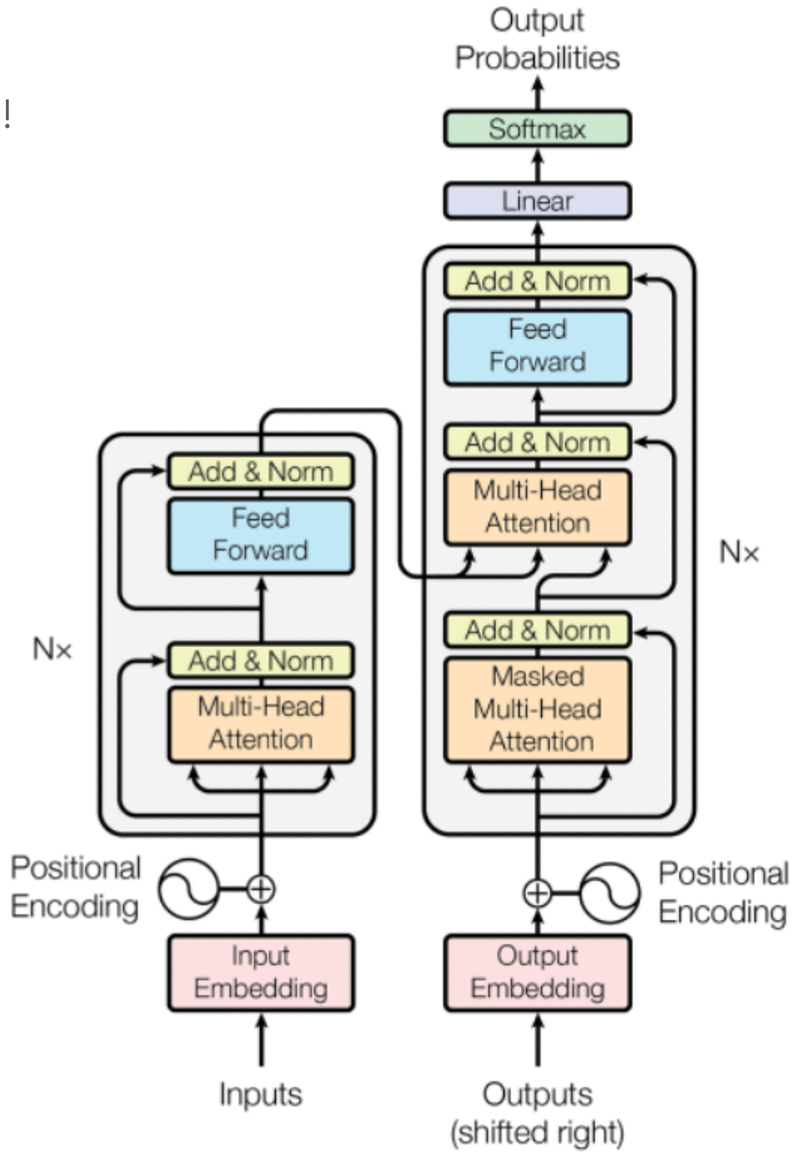
Transformers

A way to train even better models for next-word prediction!

Key intuition: a very large model with very many connections can approximate arbitrary functions.



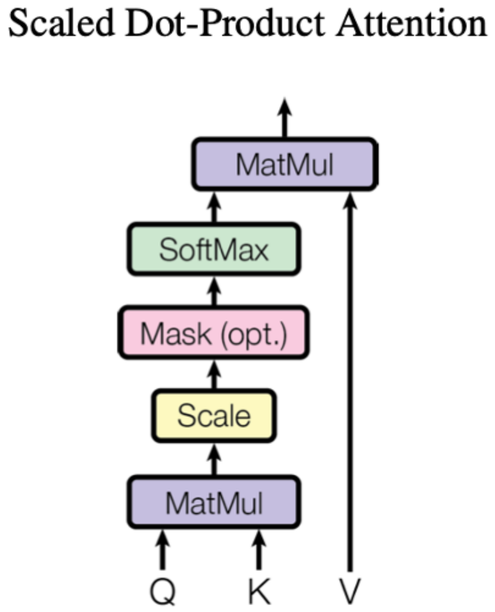


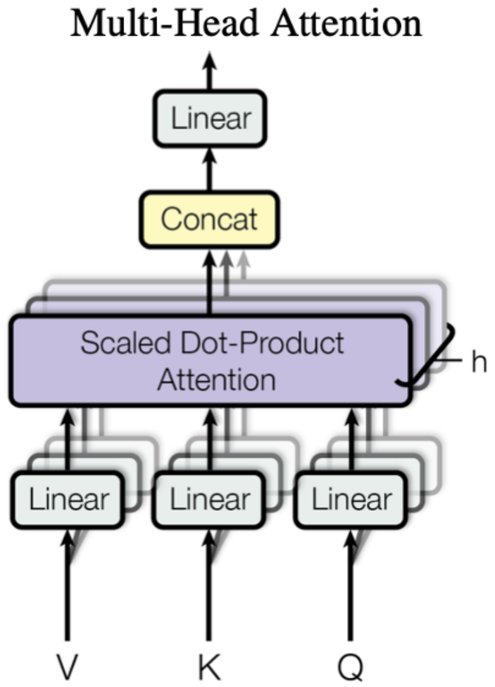


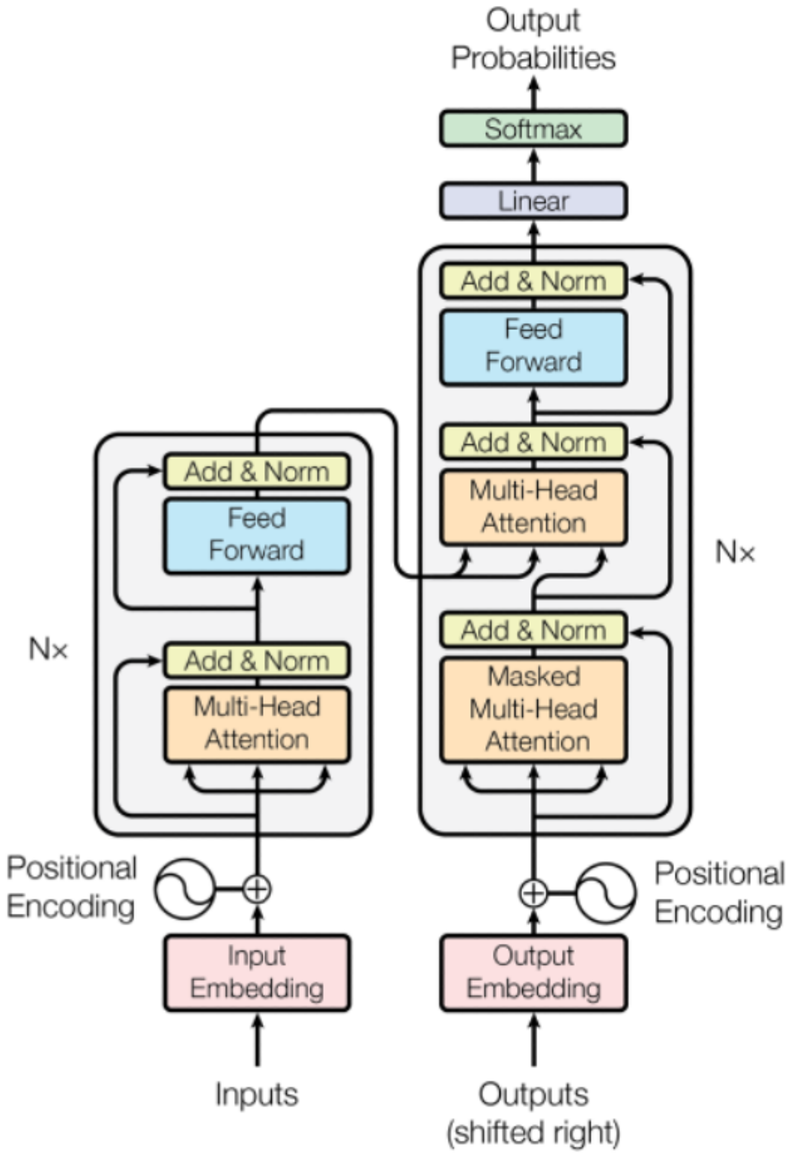
Transformers: Additional Resources

Illustrated transformer: https://jalammar.github.io/illustrated-transformer/

Annotated transformer: https://nlp.seas.harvard.edu/annotated-transformer/

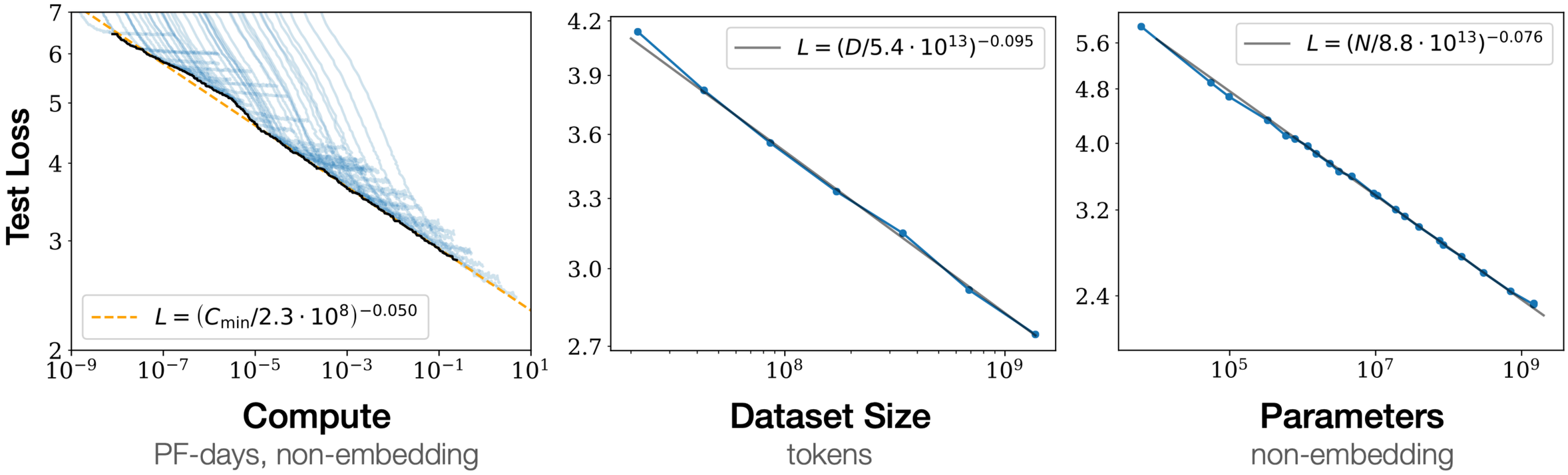






Scaling Laws for Language Models

Loss on the next-word-prediction objective is predictable as a function of training time, dataset size, and model parameters:



Next word prediction != helpful assistance

Prompt

Explain the moon landing to a 6 year old in a few sentences.

Completion

GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent
them back to the earth so we could all see them.
```

OpenAI. Aligning language models to follow instructions.

Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

Instruction tuning

We want models to be able to answer questions:

- What's the capital of France?
- What's 2 + 2?
- What's 431534 + 12341325?

Instruction tuning

We want models to be able to answer questions:

- What's the capital of France? What's the capital of Germany?
- What's 2 + 2? What's 7 +7?
- What's 431534 + 12341325? What's 3412342 * 123123?

Instruction tuning

We want models to be able to answer questions:

- What's the capital of France? Paris
- What's 2 + 2? 4
- What's 431534 + 12341325? 12,772,859

Instruction tuning

We want models to be able to answer questions:

- What's the capital of France?
- What's 2 + 2?
- What's 431534 + 12341325?

Very simple idea: train via SFT on (question, answer) pairs and mask out loss on the question.

Instruction tuning

We want models to be able to answer questions:

- What's the capital of France? Paris
- What's 2 + 2? 4
- What's 431534 + 12341325? 12,772,859

Very simple idea: train via SFT on (question, answer) pairs and mask out loss on the question.

Instruction tuning

We want models to be able to answer questions:

- What's the capital of France? Paris
- What's 2 + 2? 4
- What's 431534 + 12341325? 12,772,859

Very simple idea: train via SFT on (question, answer) pairs and mask out loss on the question. We can do this with longer-form instructions as well (e.g., "write me a story").

Chain of thought

Models aren’t very good at answering questions like “What’s 431534 + 12341325?”

Chain of thought

Models aren't very good at answering questions like "What's 431534 + 12341325?"

Instead, we can prompt them to generate intermediate reasoning before answering:

Math Word Problems (free response)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Math Word Problems (multiple choice)

Q: How many keystrokes are needed to type the numbers from 1 to 500?
Answer Choices: (a) 1156 (b) 1392 (c) 1480 (d) 1562 (e) 1788

A: There are 9 one-digit numbers from 1 to 9. There are 90 two-digit numbers from 10 to 99. There are 401 three-digit numbers from 100 to 500. 9 + 90(2) + 401(3) = 1392. The answer is (b).

CSQA (commonsense)

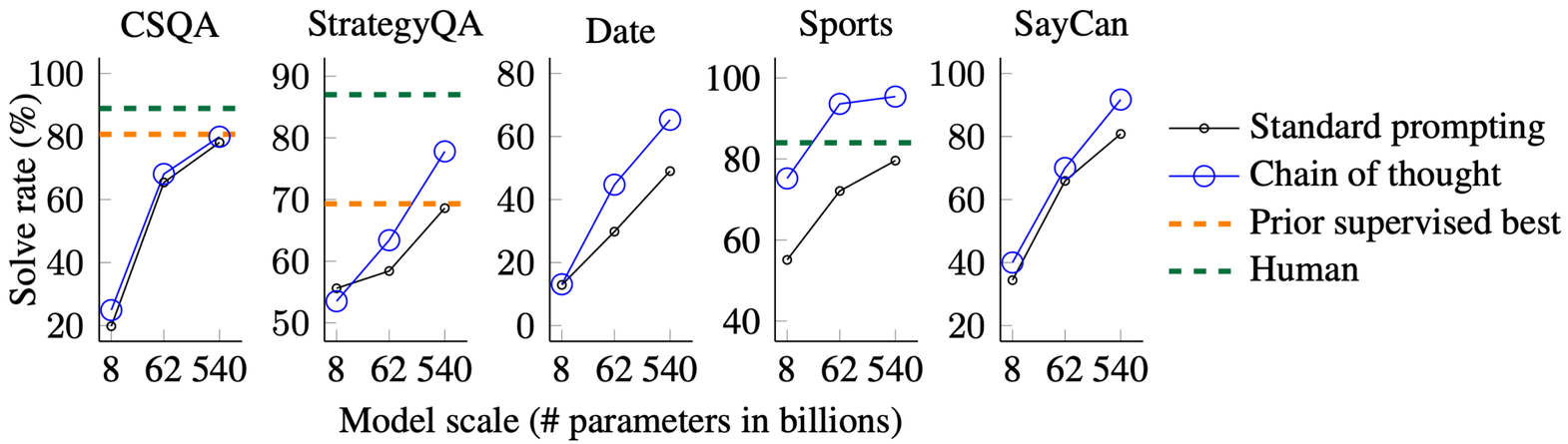
Q: Sammy wanted to go to where the people were. Where might he go?
Options: (a) race track (b) populated areas (c) desert (d) apartment (e) roadblock

A: The answer must be a place with a lot of people. Race tracks, desert, apartments, and roadblocks don't have a lot of people, but populated areas do. So the answer is (b).

Chain of thought

Models aren’t very good at answering questions like “What’s 431534 + 12341325?”

Instead, we can prompt them to generate intermediate reasoning before answering:



Chain-of-Thought Prompting Elicits Reasoning
in Large Language Models

Reasoning models: long chain of thought

Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b.

Input

Reasoning models: long chain of thought

Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b.

Input

Alright, I have this problem: Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b. First, I need to understand what 17_b and 97_b mean. The subscript b indicates that these numbers are in base b. So, 17_b and 97_b are numbers written in base b, and I need to convert them to base 10 …

Long chain of thought

Reasoning models: long chain of thought

Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b.

Input

Alright, I have this problem: Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b. First, I need to understand what 17_b and 97_b mean. The subscript b indicates that these numbers are in base b. So, 17_b and 97_b are numbers written in base b, and I need to convert them to base 10 …

Long chain of thought

The sum of all integer bases b > 9 for which 17_b is a divisor of 97_b is 70.

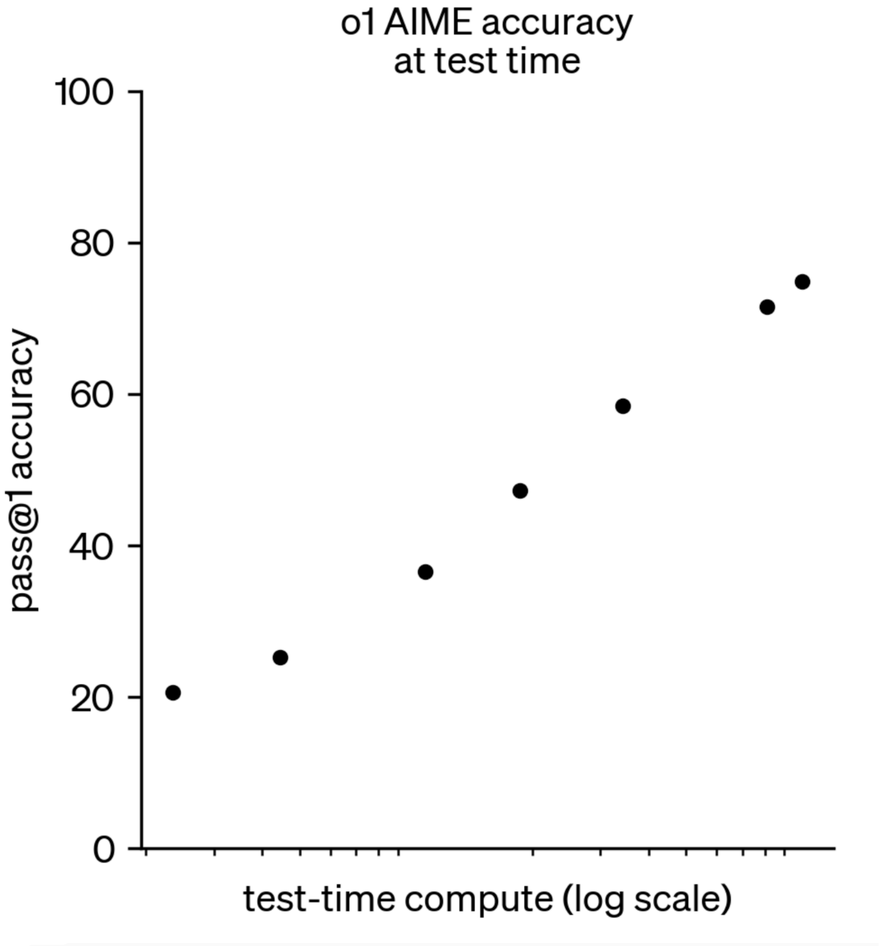
Output

Reasoning models: long chain of thought

Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b.

Alright, I have this problem: Find the sum of all integer bases b > 9 for which 17_b is a divisor of 97_b. First, I need to understand what 17_b and 97_b mean. The subscript b indicates that these numbers are in base b. So, 17_b and 97_b are numbers written in base b, and I need to convert them to base 10 …

The sum of all integer bases b > 9 for which 17_b is a divisor of 97_b is 70.



Reasoning models: long chain of thought

How to train a reasoning model (over-simplified):

Reasoning models: long chain of thought

How to train a reasoning model (over-simplified):

- Given a dataset of (question, answer) pairs, sample long CoTs from model

Reasoning models: long chain of thought

How to train a reasoning model (over-simplified):

- Given a dataset of (question, answer) pairs, sample long CoTs from model
- Select the subset of CoTs that lead to the correct answer

Reasoning models: long chain of thought

How to train a reasoning model (over-simplified):

- Given a dataset of (question, answer) pairs, sample long CoTs from model
- Select the subset of CoTs that lead to the correct answer
- Train the model via SFT on (question, long CoT, answer) sequences from this filtered set

Reasoning models: long chain of thought

How to train a reasoning model (over-simplified):

- Given a dataset of (question, answer) pairs, sample long CoTs from model
- Select the subset of CoTs that lead to the correct answer
- Train the model via SFT on (question, long CoT, answer) sequences from this filtered set

This is **filtered behavior cloning**.

Reasoning models: long chain of thought

In practice, most reasoning models use GRPO instead of filtered BC:

- For each (question, answer) pair, generate a group of trajectories
- Score each trajectory relative to the baseline performance of the group
- Assign positive reward to each token in above-average trajectories, and negative otherwise

Reasoning models: long chain of thought

In practice, most reasoning models use GRPO instead of filtered BC:

- For each (question, answer) pair, generate a group of trajectories
- Score each trajectory relative to the baseline performance of the group
- Assign positive reward to each token in above-average trajectories, and negative otherwise

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}\hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right] - \beta\mathbb{D}_{KL}\left[\pi_\theta||\pi_{ref}\right]\right\},$$

Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

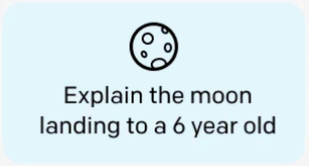
Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

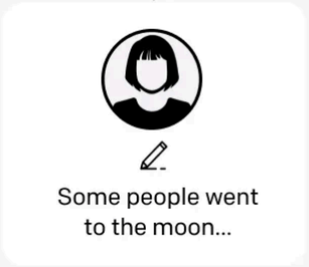
Step 1

Collect demonstration data, and train a supervised policy.

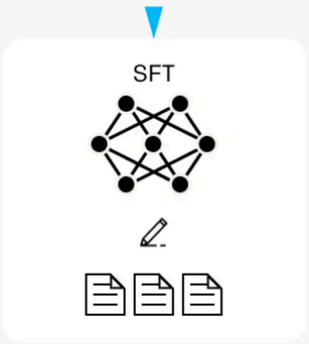
A prompt is sample from our prompt dataset.



A labeler demonstrates the desired output behavior.



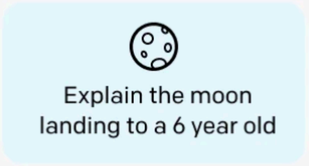
This data is used to fine-tune GPT-3 with supervised learning.



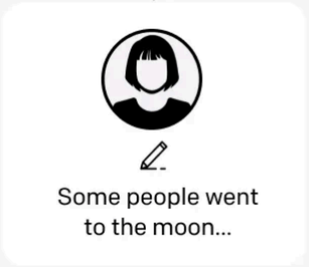
Step 1

Collect demonstration data, and train a supervised policy.

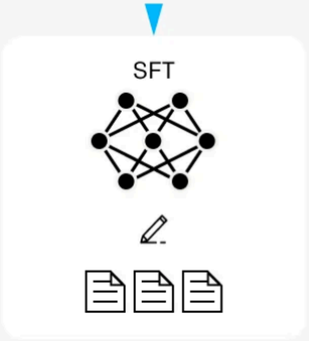
A prompt is sample from our prompt dataset.



A labeler demonstrates the desired output behavior.



This data is used to fine-tune GPT-3 with supervised learning.



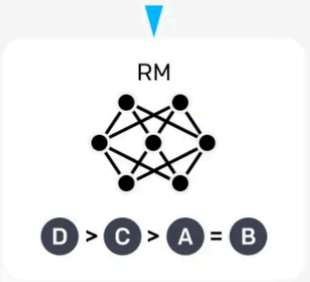
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

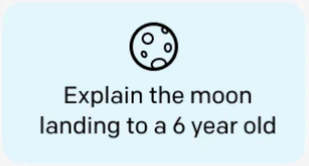
This data is used to train our reward model.



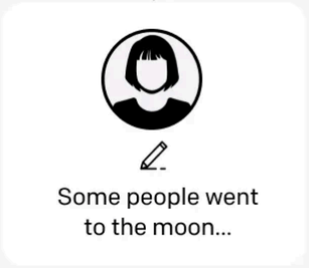
Step 1

Collect demonstration data, and train a supervised policy.

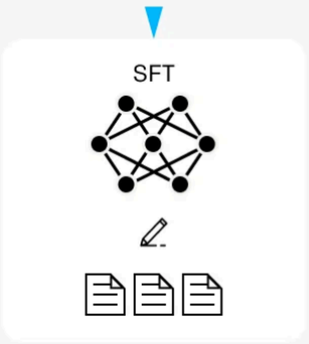
A prompt is sample from our prompt dataset.



A labeler demonstrates the desired output behavior.



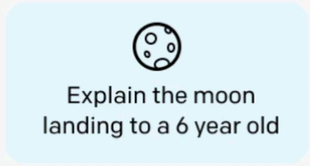
This data is used to fine-tune GPT-3 with supervised learning.



Step 2

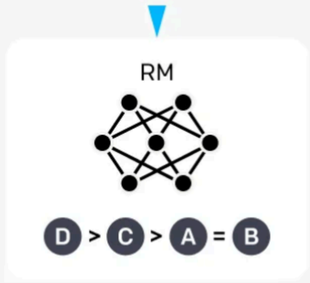
Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



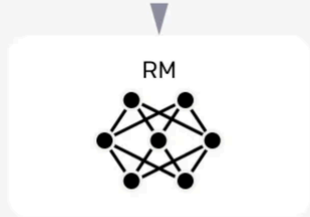
Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.

Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

Outline

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

“Don’t teach, incentivize”

- RLHF: don’t just train models on helpful examples. Instead, train a reward model first, and then optimize the reward model.
- RLVR: don’t just train on solution traces. Instead, find solutions which optimize the probability of generating the correct answer.

“Don’t teach, incentivize”

- RLHF: don’t just train models on helpful examples. Instead, train a reward model first, and then optimize the reward model.
- RLVR: don’t just train on solution traces. Instead, find solutions which optimize the probability of generating the correct answer.

What’s the analogue for multi-turn interaction?

A simple recipe for building interactive models

Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple recipe for building interactive models

Step 1: Collect simulated dialogues

Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple recipe for building interactive models

Step 1: Collect simulated dialogues

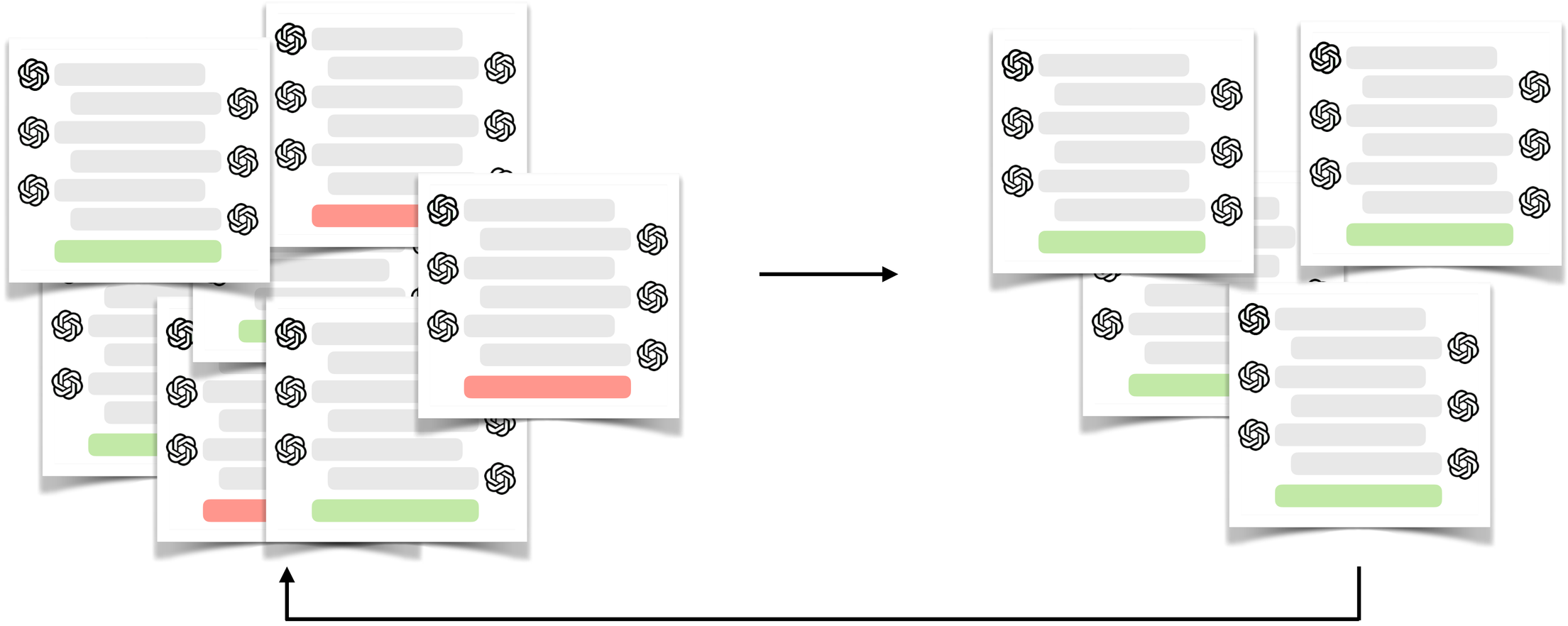
Step 2: Finetune on above-average dialogues

Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple recipe for building interactive models

Step 1: Collect simulated dialogues

Step 2: Finetune on above-average dialogues



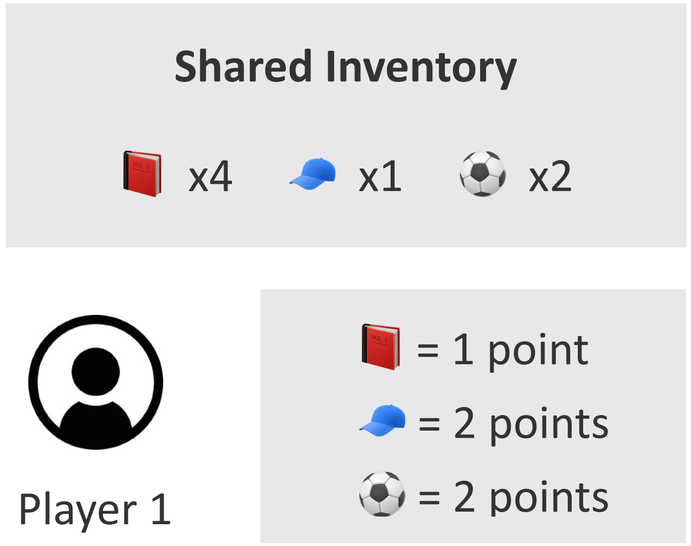
Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple testbed for language model self-play

Shared Inventory

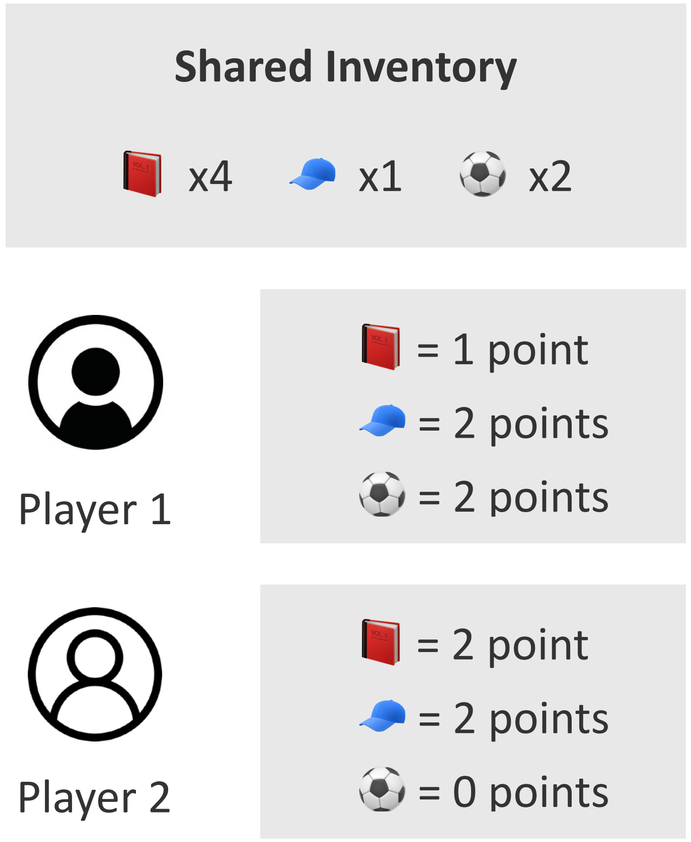
Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play



Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play



Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play

Shared Inventory

📕 x4 🧢 x1 ⚽ x2

Player 1

📕 = 1 point
🧢 = 2 points
⚽ = 2 points

Player 2

📕 = 2 point
🧢 = 2 points
⚽ = 0 points

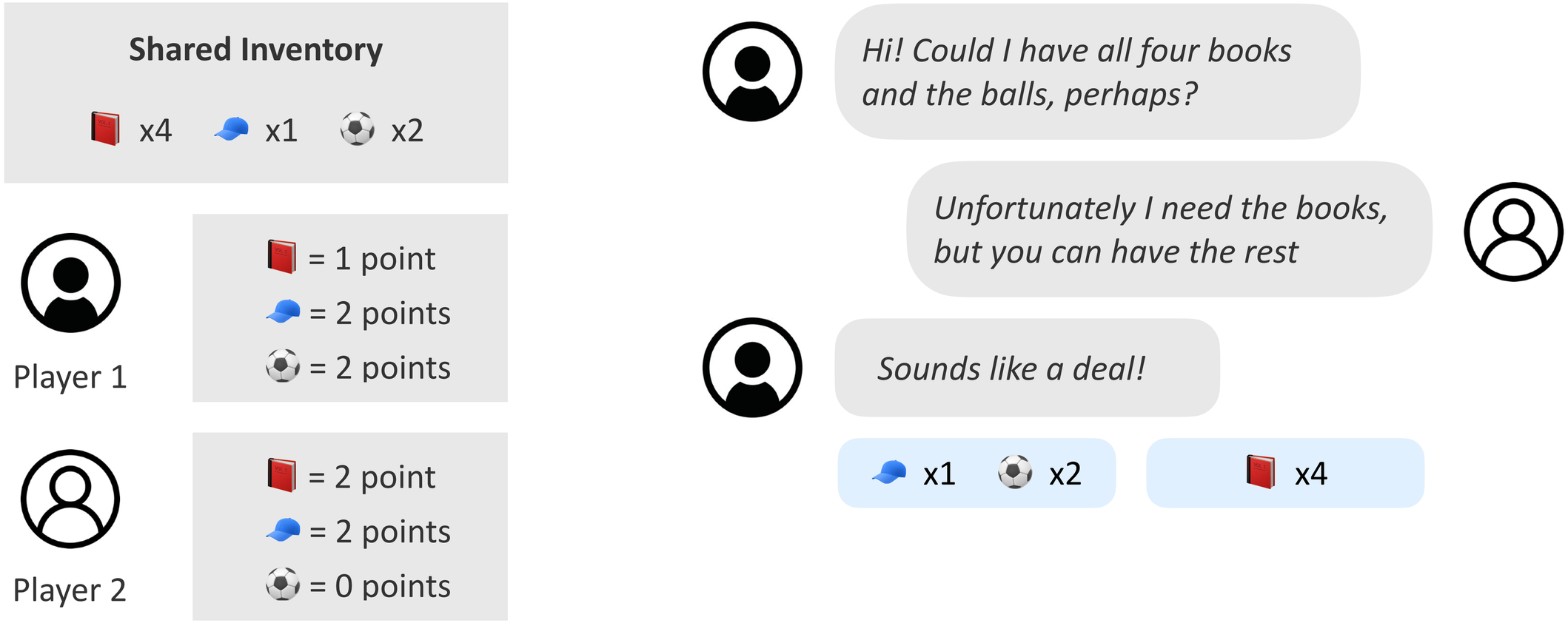
Hi! Could I have all four books and the balls, perhaps?

Unfortunately I need the books, but you can have the rest

Sounds like a deal!

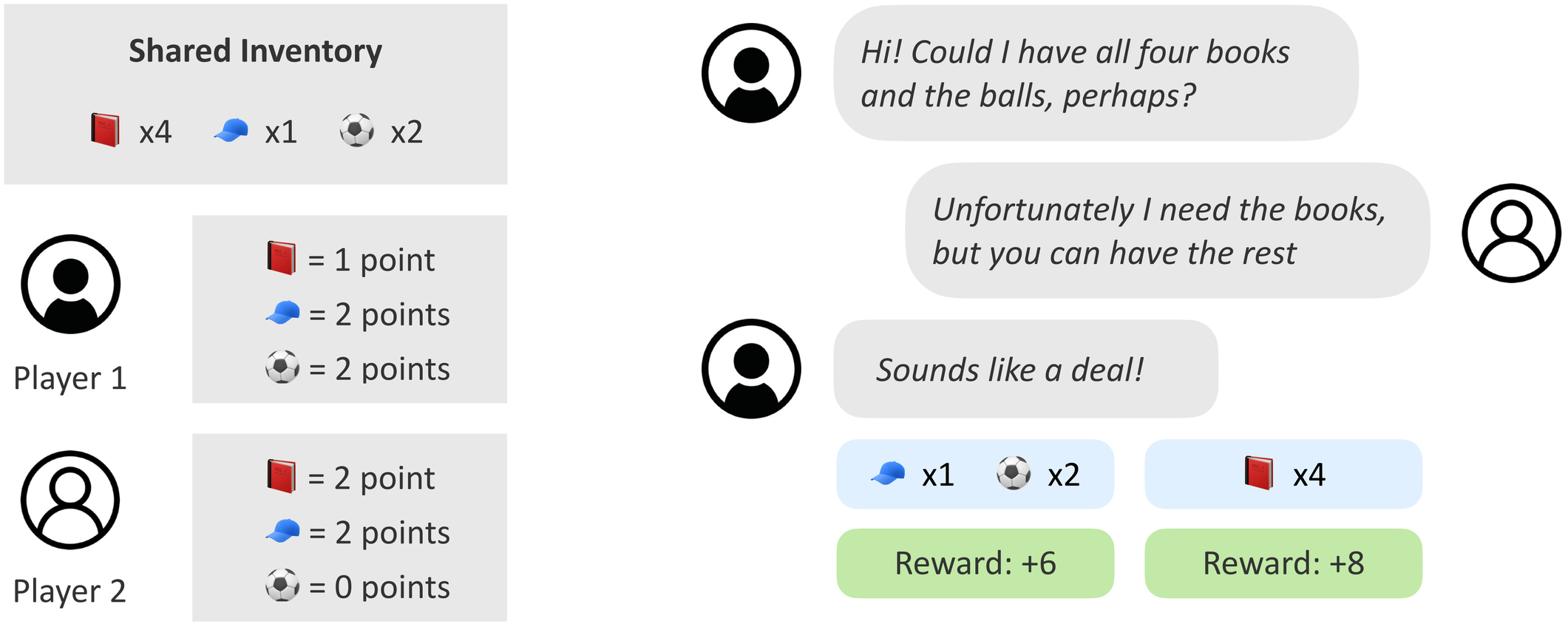
Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play



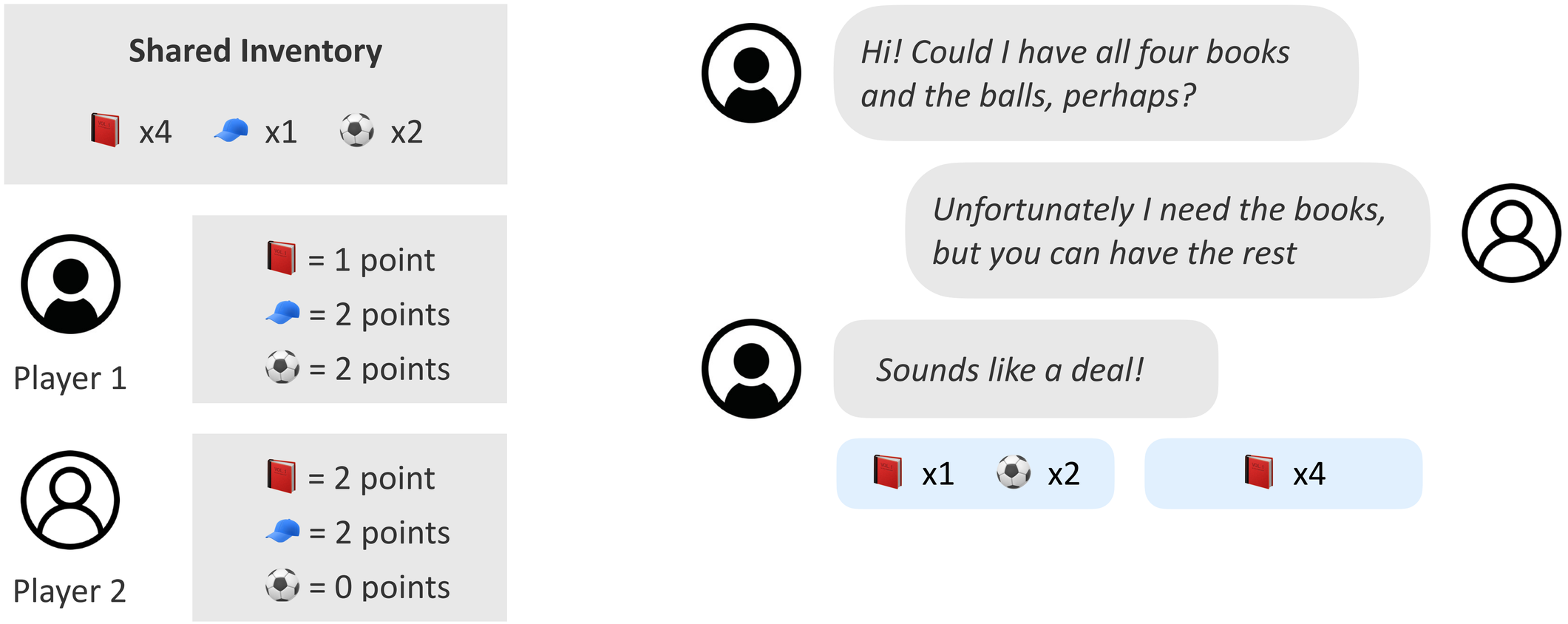
Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play



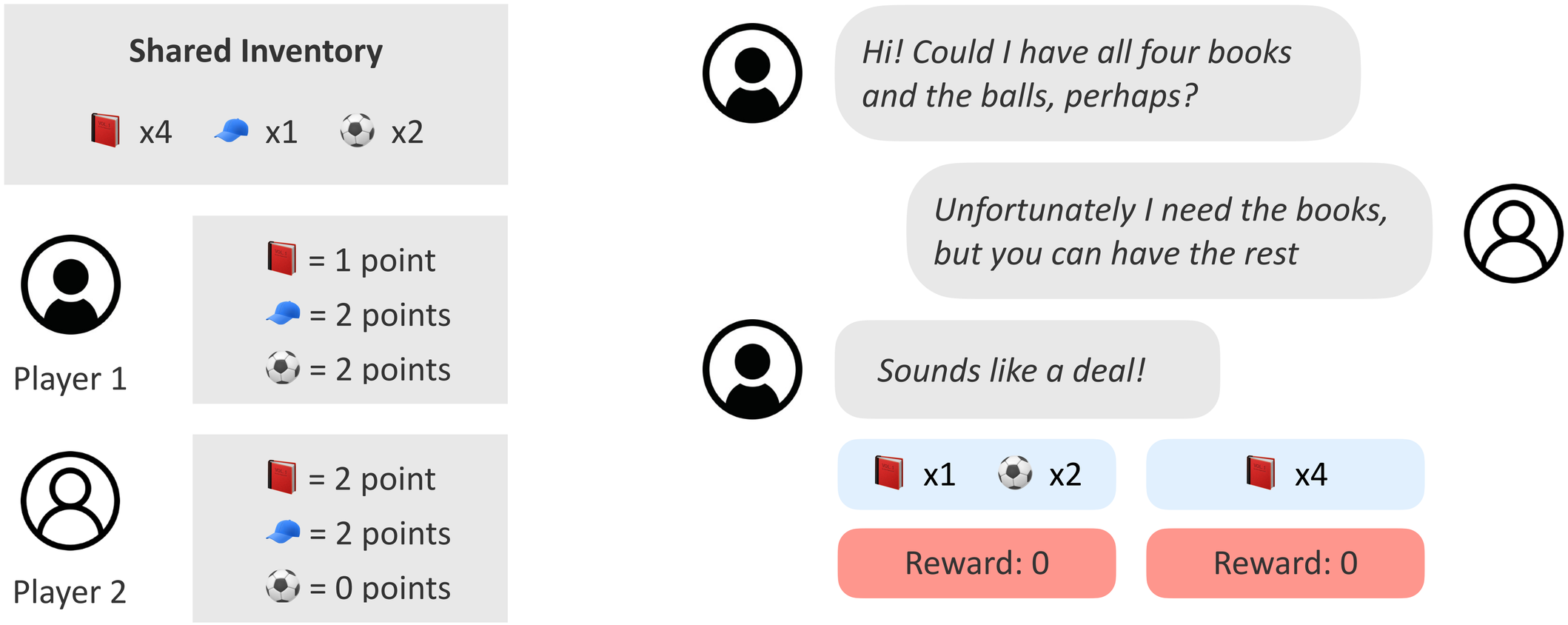
Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play



Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple testbed for language model self-play

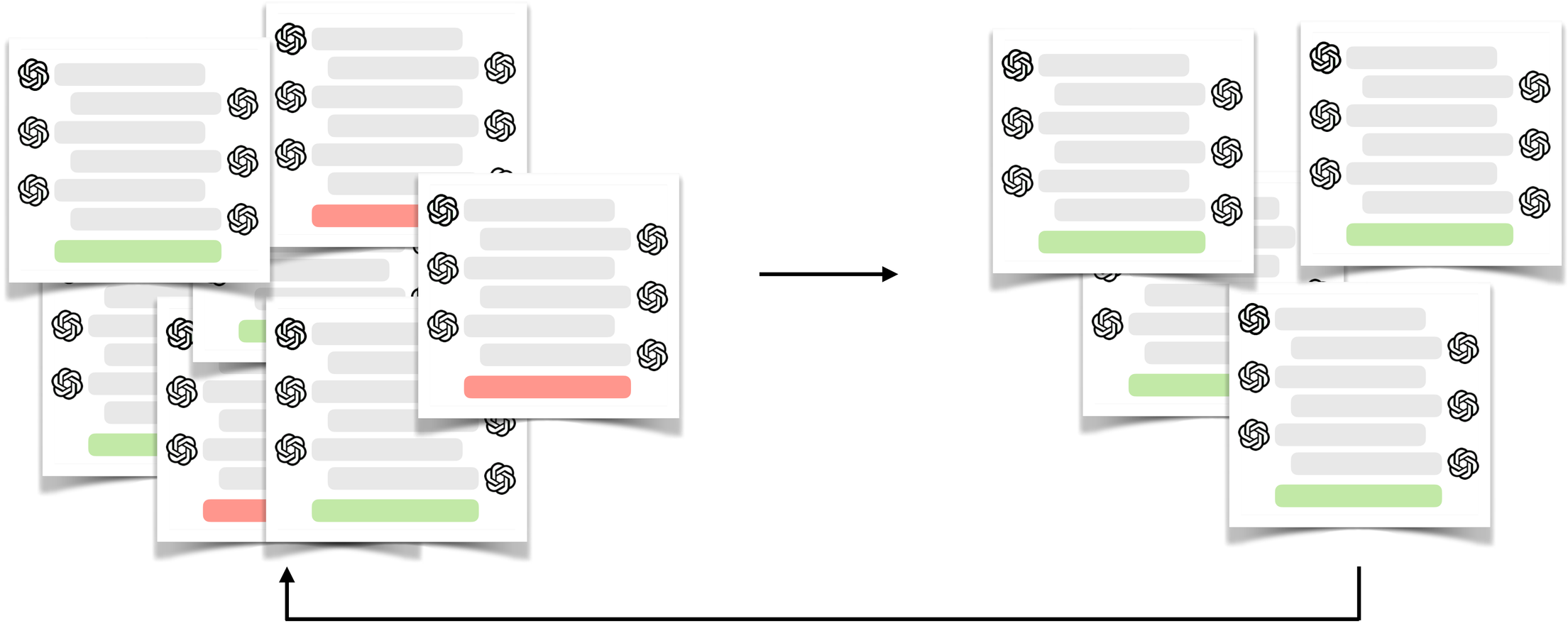


Deal or no deal? End-to-end learning for negotiation dialogues. Mike Lewis, Denis Yarats, Yann N. Dauphin, Devi Parikh, Dhruv Batra. EMNLP 2017.

A simple recipe for building interactive models

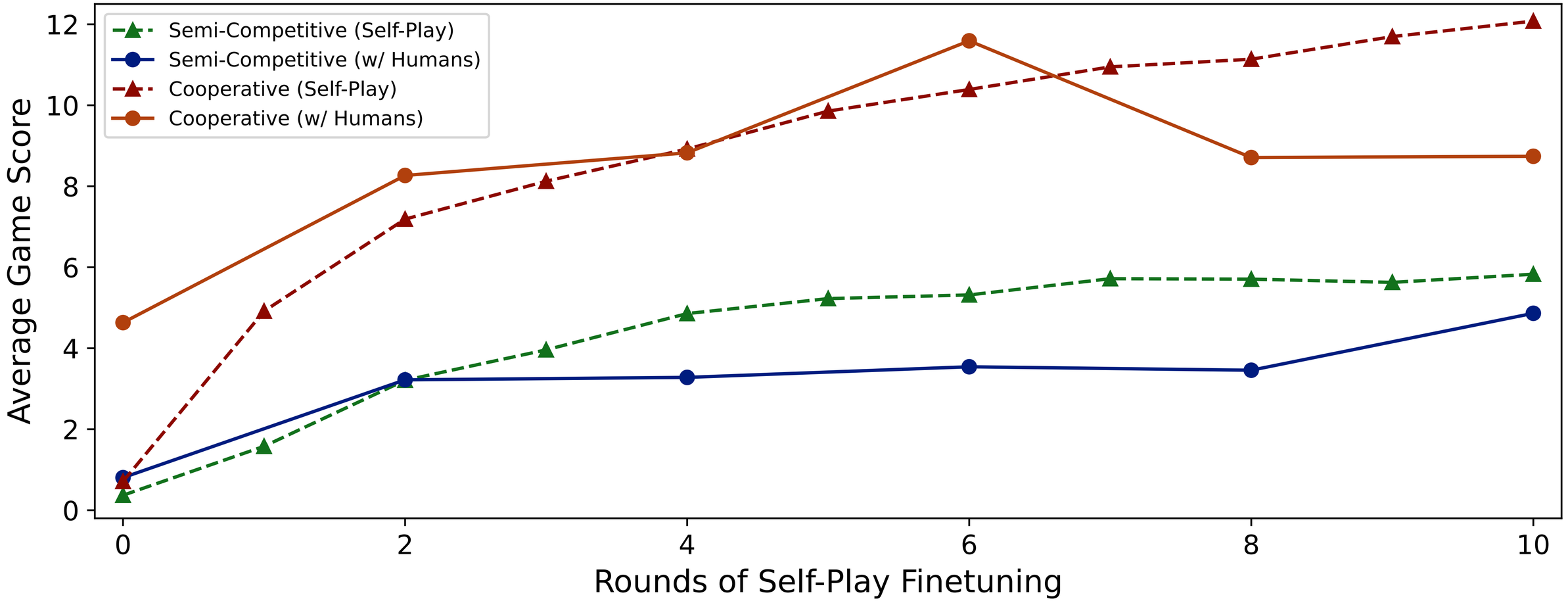
Step 1: Collect simulated dialogues

Step 2: Finetune on above-average dialogues



Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

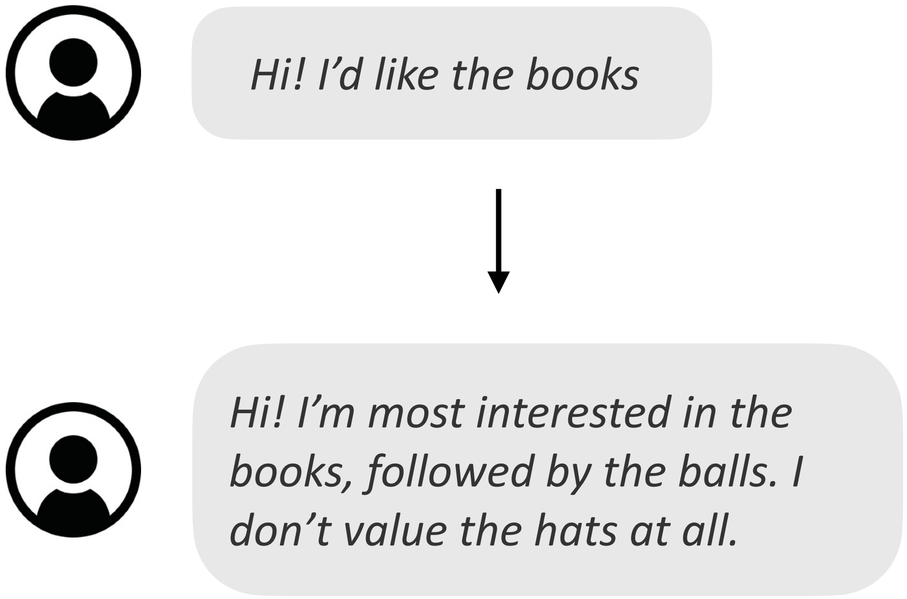
A simple testbed for language model self-play



Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple testbed for language model self-play

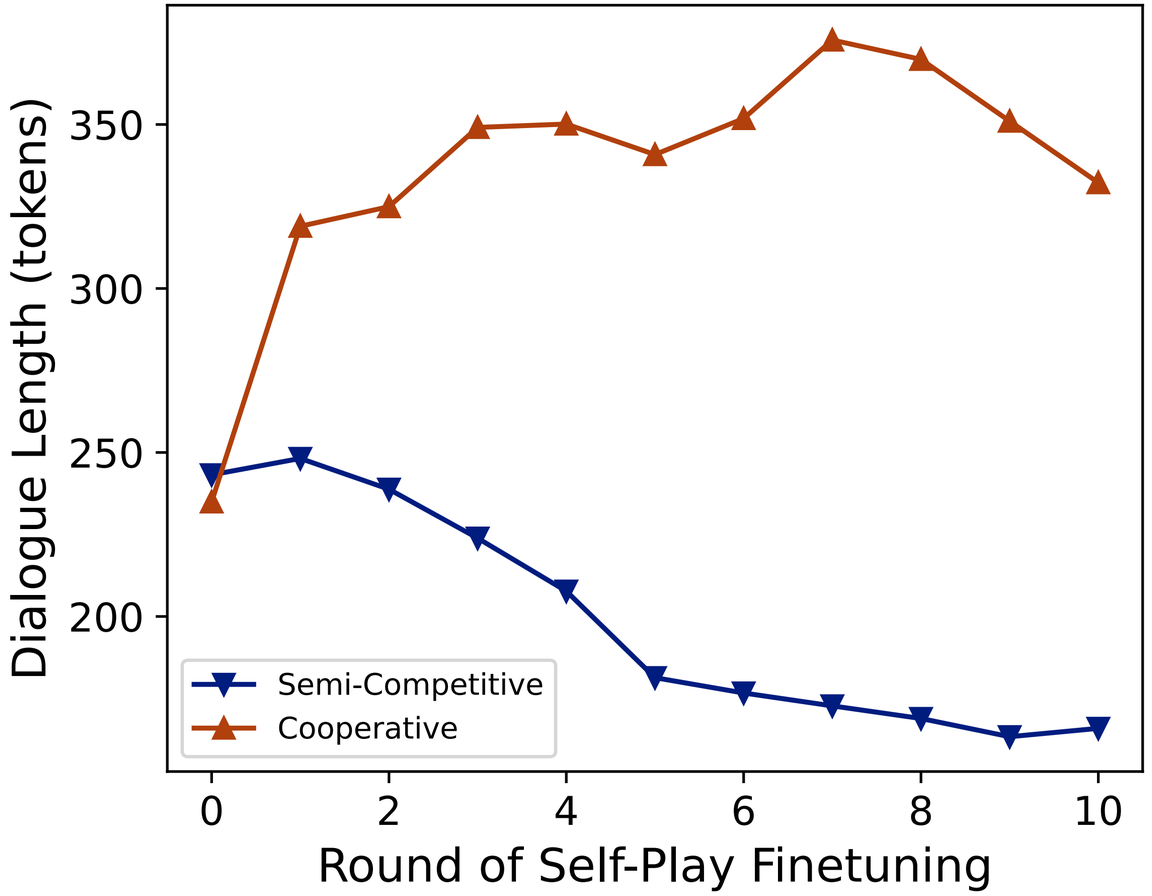
- Increase in percent of private values shared
- Increase in specificity of information shared



Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple testbed for language model self-play

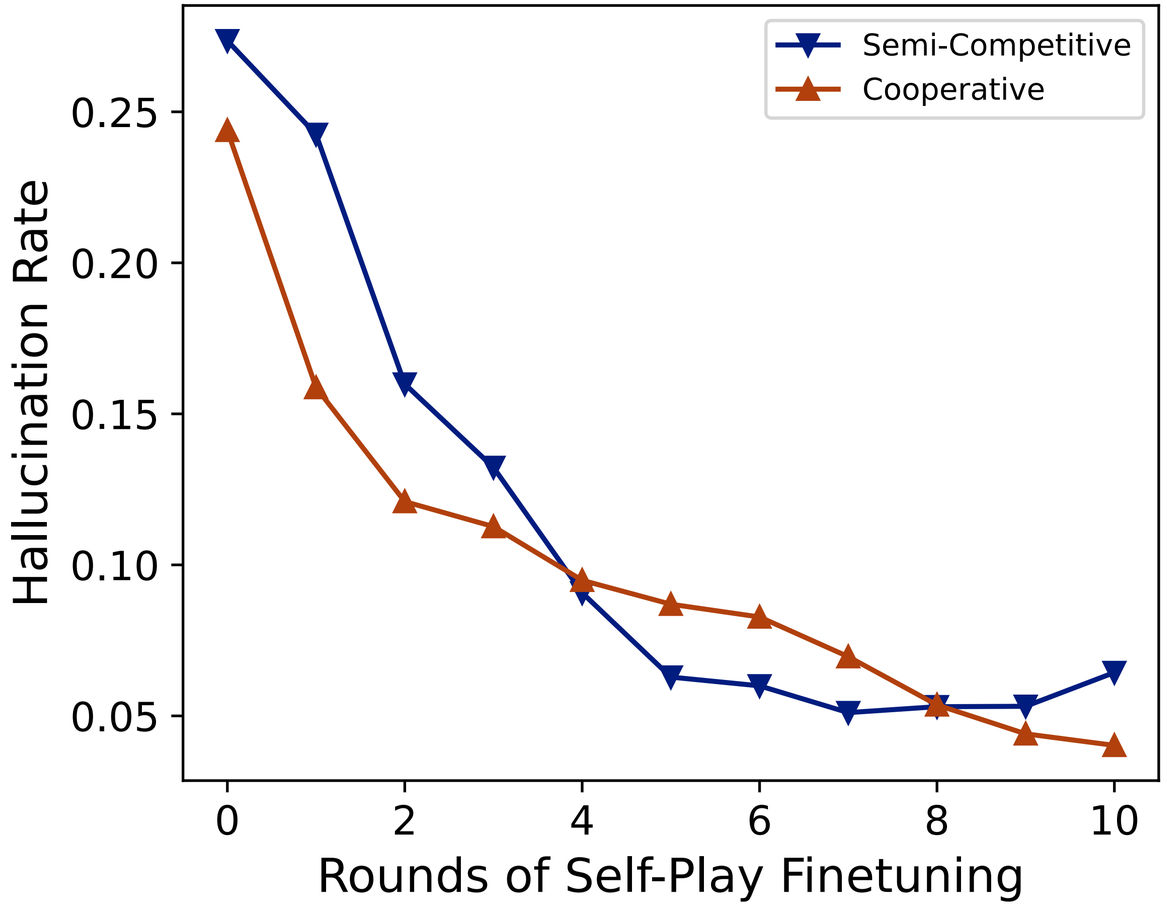
- Increase in percent of private values shared
- Increase in specificity of information shared



Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

A simple testbed for language model self-play

- Increase in task understanding and instruction following
- Decrease in number of messages containing incorrect private values



Efficacy of language model self-play in non-zero-sum games. Austen Liao*, **Nicholas Tomlin***, Dan Klein. Preprint, 2024.

Recap: Objectives for LLM Training

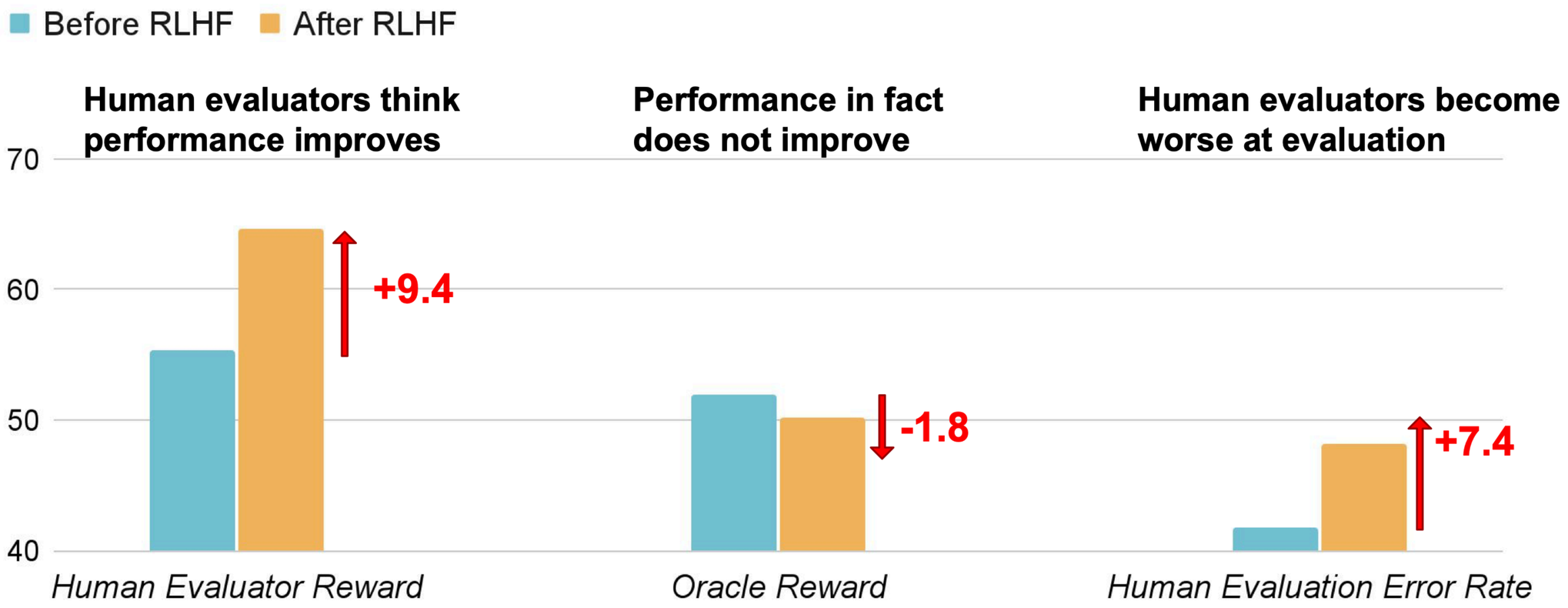
- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

Recap: Objectives for LLM Training

- Next token prediction
- Answer correctness (reinforcement learning from verifiable rewards)
- Human preference (reinforcement learning from human feedback)
- Collaborative success

Many different possible objectives, it's important to optimize for the right ones!

What happens when objectives go wrong?



Language models learn to mislead humans via RLHF. Wen, et al. 2024

What happens when objectives go wrong?

- We know that models sometimes hallucinate, but hallucinations are generally declining these days

- However, if our reward models (e.g., human preferences) prefer answers that contain hallucinations, then models will continue hallucinating, and will do so in harder-to-detect ways

Investigating truthfulness in a pre-release o3 model. Chowdhury, et al. 2025

What happens when objectives go wrong?

- We know that models sometimes hallucinate, but hallucinations are generally declining these days
- However, if our reward models (e.g., human preferences) prefer answers that contain hallucinations, then models will continue hallucinating, and will do so in harder-to-detect ways
- **These sorts of errors have real consequences; think about the societal impact of the systems you build!**

Investigating truthfulness in a pre-release o3 model. Chowdhury, et al. 2025

That’s all folks!

- Thanks for being a part of this class this semester!
- This is our first time teaching the material, let us know if there are things we can do better next time
- Projects are due on May 8th, 11:59PM ET
- We’ll try our best to get final grades back ASAP

Course Evaluations