

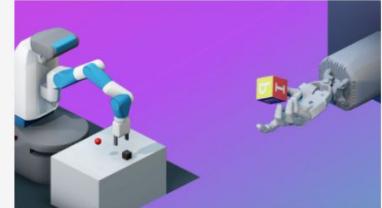
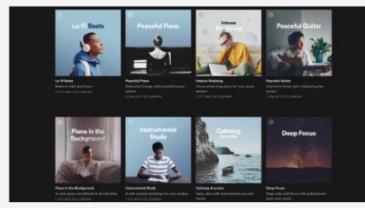
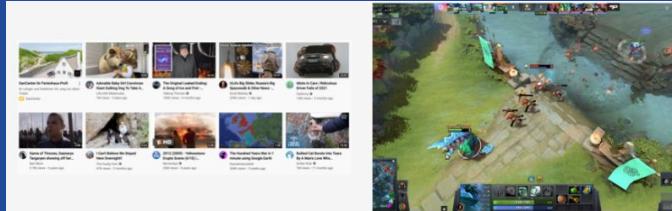
# Reinforcement Learning for Recommender Systems

From Contextual Bandits to Slate-Q

---

Sven Mika, PhD

<https://linkedin.com/in/sven-mika>  
sven@anyscale.com





# Who is the RL Team @ Anyscale?

<https://docs.ray.io>



**Sven**



**Jun**



**Avnish**



**Artur**



**Steven**



**Christy**  
**(Developer Advocate)**

**Summer '22:**

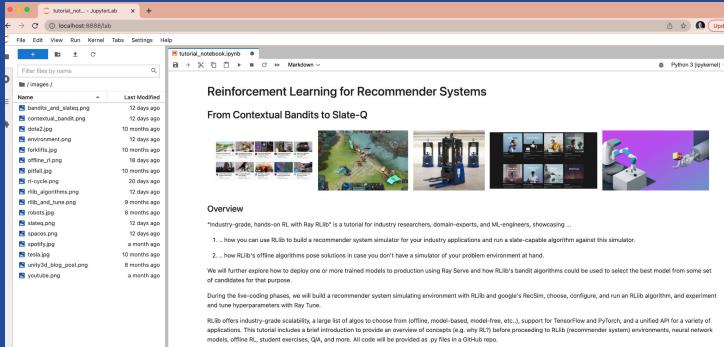
**Kourosh**

**Justin**

# Overview of the tutorial

## Our iPython Notebook

... and how to git it



```
$ git clone https://github.com/sven1977/rllib\_tutorials
$ cd rllib_tutorials/production_rl_summit_2022
$ jupyter-lab
```

---

```
$ git clone https://github.com/sven1977/rllib\_tutorials
$ cd rllib_tutorials/production_rl_summit_2022
$ jupyter-lab
```

# Overview of the tutorial

30 min: Intro to RL - Why should we use RL to solve Recommender Systems?

15 min: break

30 min: RLlib's Contextual Bandits and SlateQ on Google RecSim problem

15 min: break

30 min: Offline RL; RLlib + Ray Serve for policy production deployment

---

**~2.0h**

---

```
$ git clone https://github.com/sven1977/rllib\_tutorials
$ cd rllib_tutorials/production_rl_summit_2022
$ jupyter-lab
```



# Recommender Systems (Personalization Systems)

All Seminars Machine learning Tiny Desk Concerts Conversation Live Music San Francisco Symphony Hiking Violins APIs Jazz Wood Nature Rapping Art

Jeremy Howard interviews Sanyam Bhutani  
Special Episode

53:57

Jeremy Howard Interviews Kaggle Grandmaster Sanyam Bhutani  
Jeremy Howard  
2.6K views • 1 day ago

Track Sprint Workout: How To  
Jade Teta

24K views • 10 years ago

Anderson .Paak & The Free Nationals:  
NPR Music Tiny Desk Concert

NPR Music

85M views • 5 years ago

I AAAI 20 / AAAI 2020 Keynotes Turing Award Winners Event / Geoff Hinton,..

ICML IJCAI ECAI 2018 Conference Videos

37K views • 2 years ago

LIVE  
BREAKING NEWS  
Russia invades Ukraine

DW

Russia invades Ukraine LIVE | DW News  
livestream | Headline news from aroun...

DW News

23K watching

LIVE NOW

phantom Thread - House of Woodcock (Official Audio)

Nonesuch Records

1.3M views • 4 years ago

think Think Fast, Talk Smart: Communication Techniques

Stanford Graduate School of Business

25M views • 7 years ago

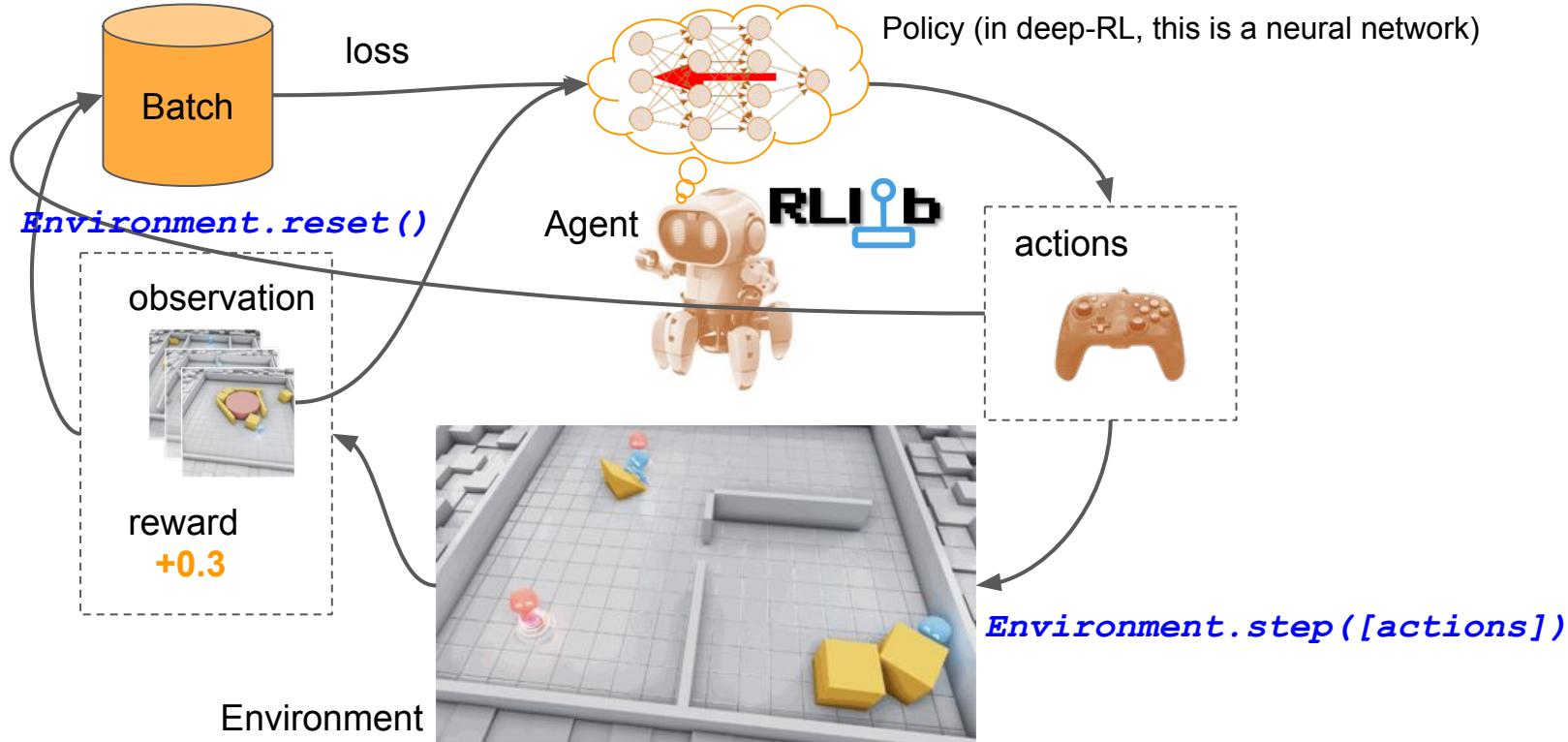
Hiking 60 Miles Alone in Hornstrandir Iceland

Kraig Adams

5M views • 2 years ago



# What's Reinforcement Learning (RL)?

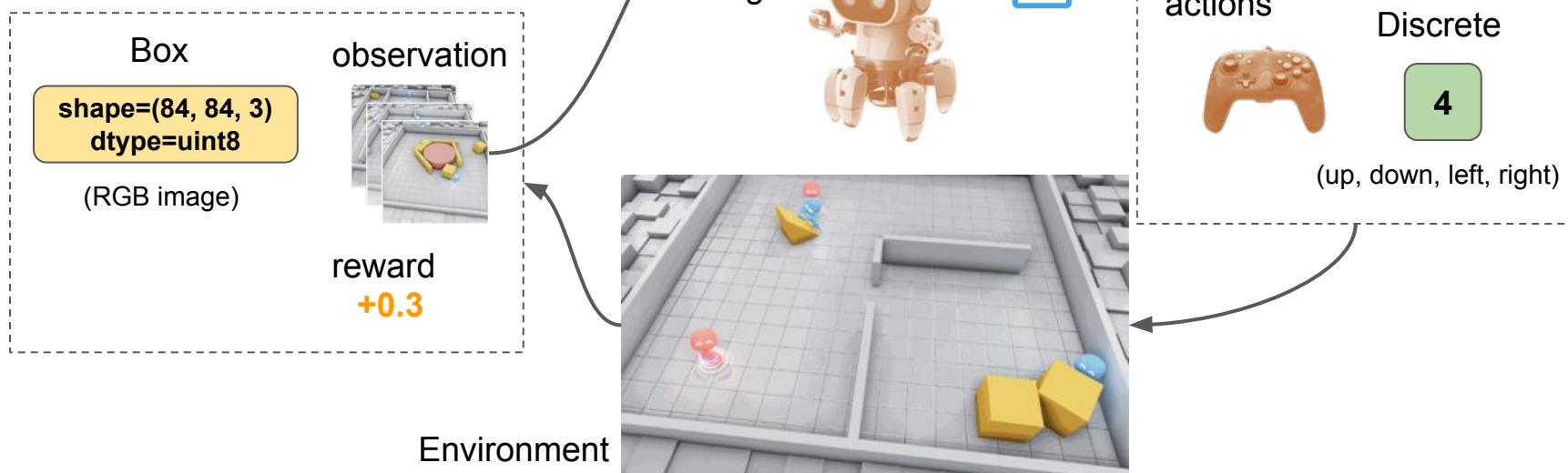


```
$ git clone https://github.com/sven1977/rllib_tutorials  
$ cd rllib_tutorials/production_rl_summit_2022  
$ jupyter-lab
```



# What's Reinforcement Learning (RL)?

## Gaming

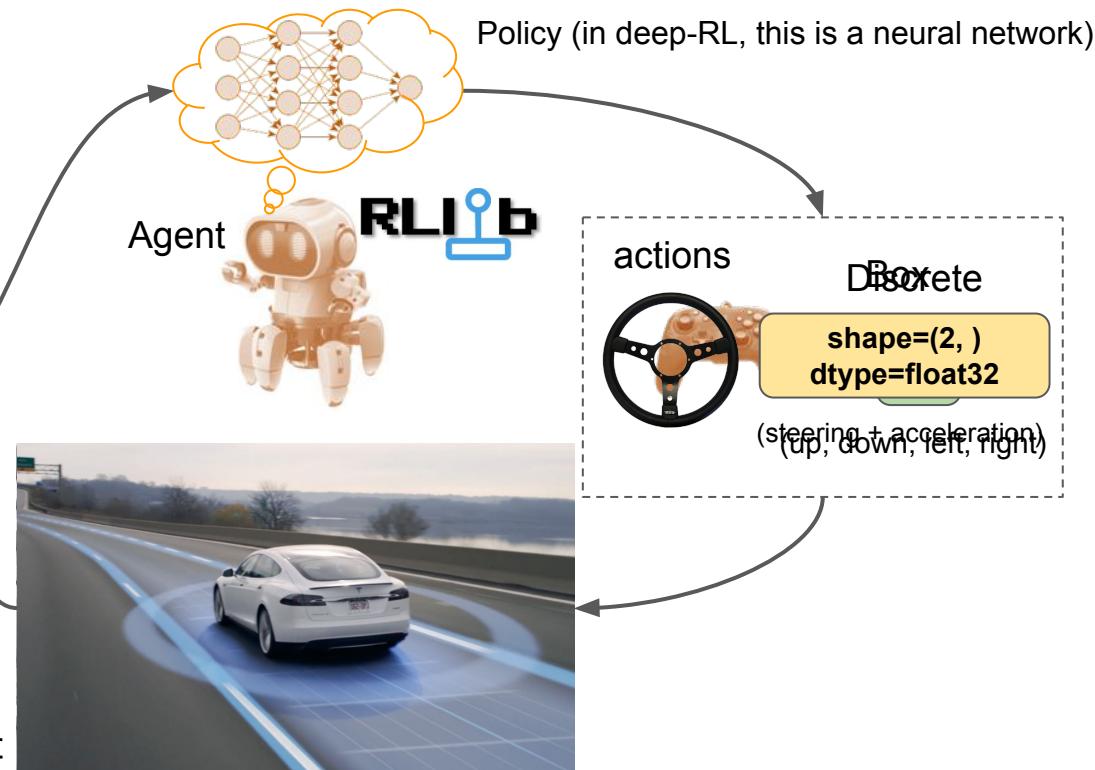
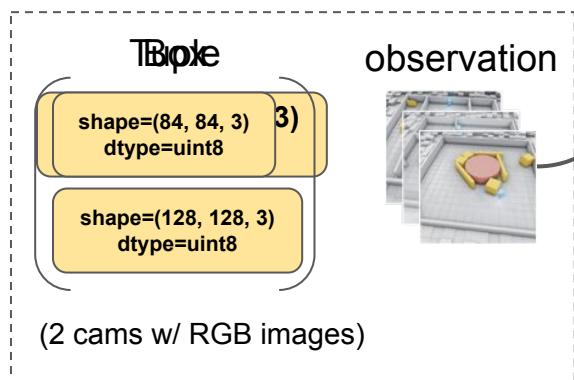


```
$ git clone https://github.com/sven1977/rllib_tutorials  
$ cd rllib_tutorials/production_rl_summit_2022  
$ jupyter-lab
```



# What's Reinforcement Learning (RL)?

## Self-Driving

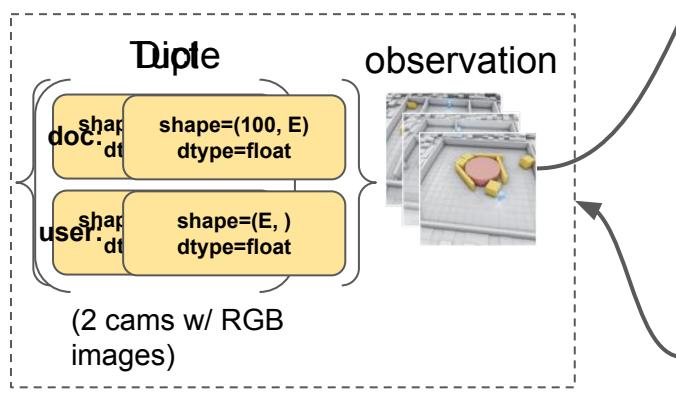


```
$ git clone https://github.com/sven1977/rllib_tutorials  
$ cd rllib_tutorials/production_rl_summit_2022  
$ jupyter-lab
```

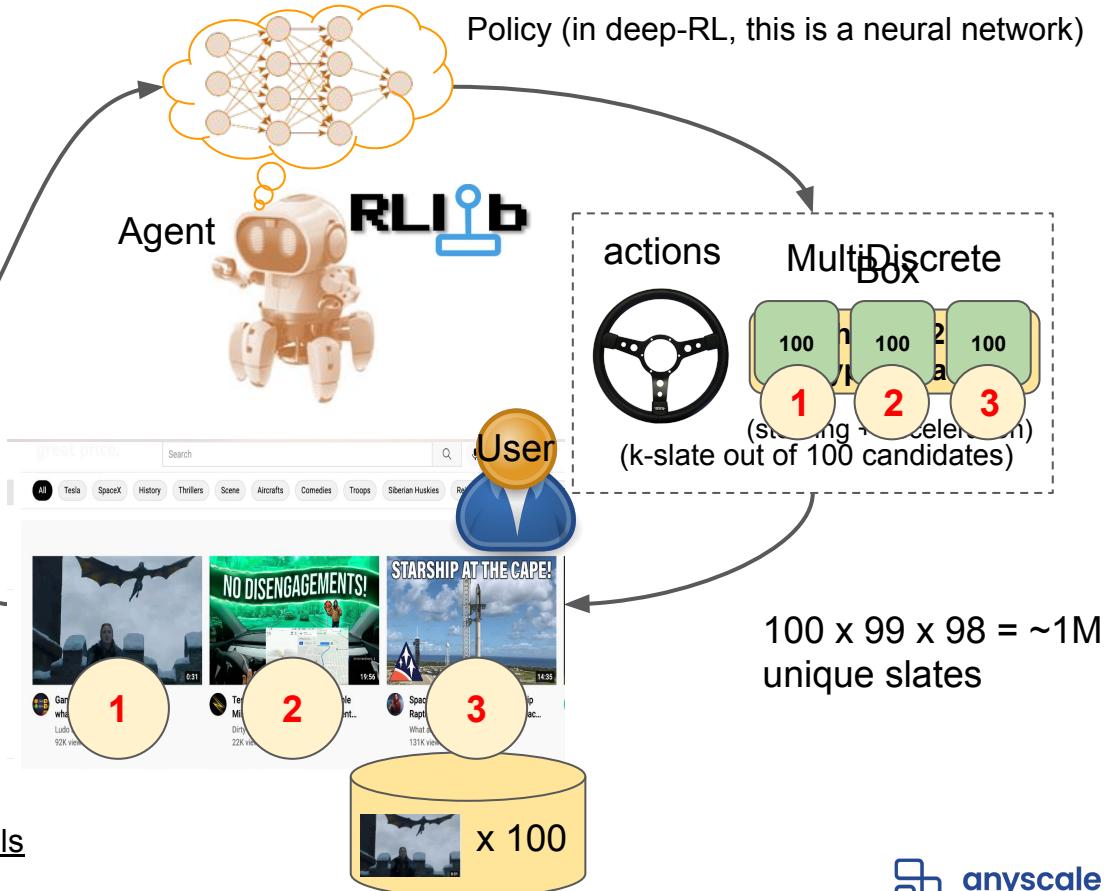


# What's Reinforcement Learning (RL)?

## Recommender Systems



Environment

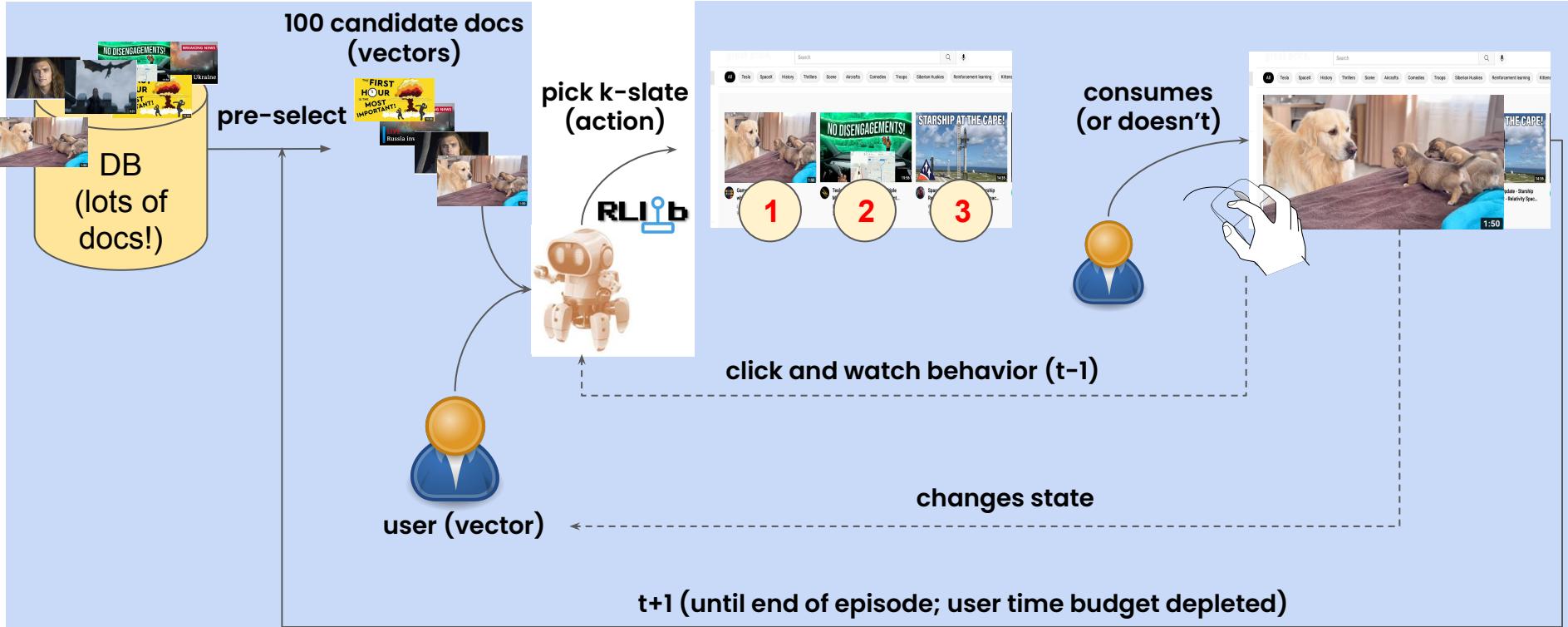


```
$ git clone https://github.com/sven1977/rllib_tutorials  
$ cd rllib_tutorials/production_rl_summit_2022  
$ jupyter-lab
```





# A Recommender System in Action



```
$ git clone https://github.com/sven1977/rllib_tutorials  
$ cd rllib_tutorials/production_rl_summit_2022  
$ jupyter-lab
```

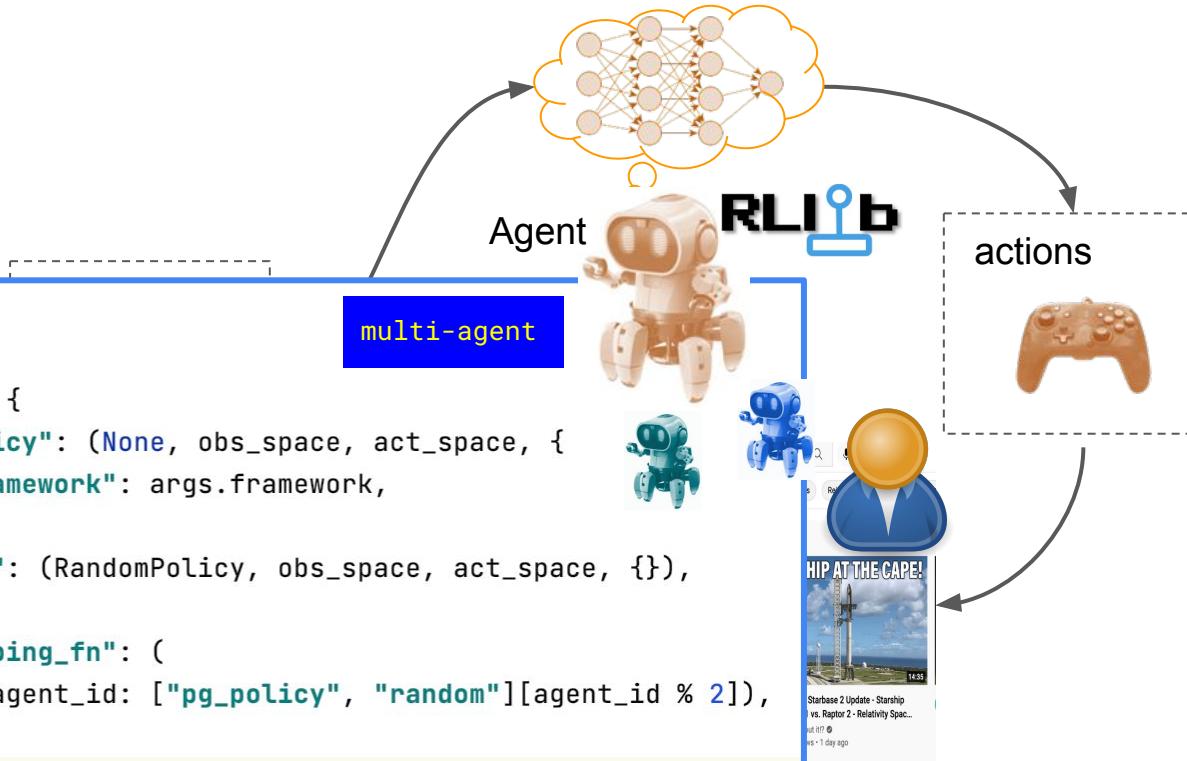


# And why should I use RLlib?

```
config = {  
    "multiagent": {  
        "policies": {  
            "pg_policy": (None, obs_space, act_space, {  
                "framework": args.framework,  
            }),  
            "random": (RandomPolicy, obs_space, act_space, {}),  
        },  
        "policy_mapping_fn": (  
            lambda agent_id: ["pg_policy", "random"][agent_id % 2]),  
    },  
}
```

```
results = tune.run("PG", config=config, stop=stop, verbose=1)
```

\$ jupyter-lab





# And why should I use RLlib?

TensorFlow &  
PyTorch Policies

```
class CustomModel(TFModelV2):
    """Example of a keras custom model that just delegates to an fc-net."""

    def __init__(self, obs_space, action_space, num_outputs, model_config,
                 name):
        super(CustomModel, self).__init__(obs_space, action_space, num_outputs,
                                         model_config, name)
        self.model = FullyConnectedNetwork(obs_space, action_space,
                                           num_outputs, model_config, name)

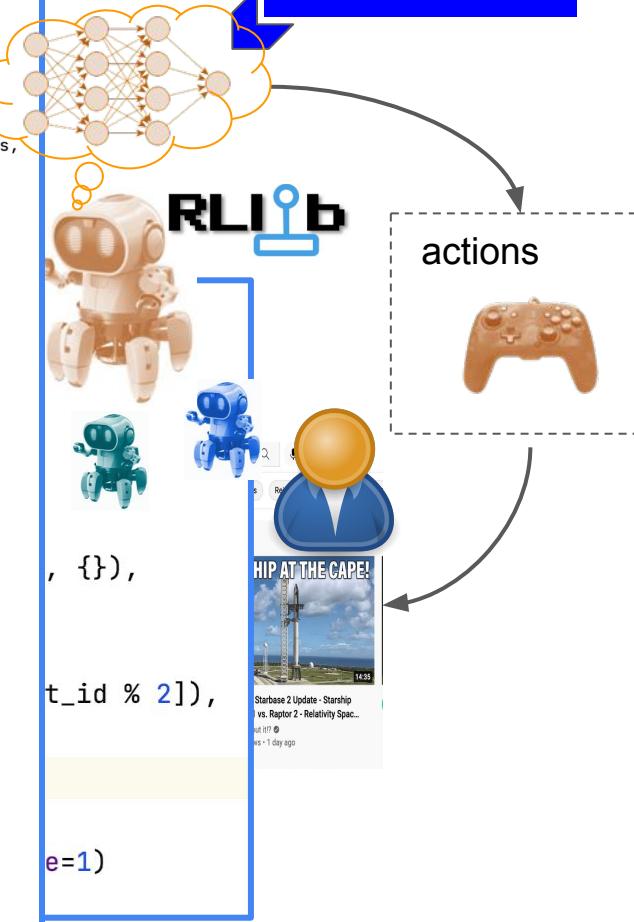
    def forward(self, input_dict, state, seq_lens):
        return self.model.forward(input_dict, state, seq_lens)

    def value_function(self):
        ModelCatalog.register_custom_model(
            "my_model", TorchCustomModel
            if args.framework == "torch" else CustomModel)

        config = {
            "model": {
                "custom_model": "my_model",
            },
        }

        def forward(self, input_dict, state, seq_lens):
            input_dict["obs"] = input_dict["obs"].float()
            fc_out, _ = self.torch_sub_model(input_dict, state, seq_lens)
            return fc_out, []

        def value_function(self):
            return torch.reshape(self.torch_sub_model.value_function(), [-1])
```



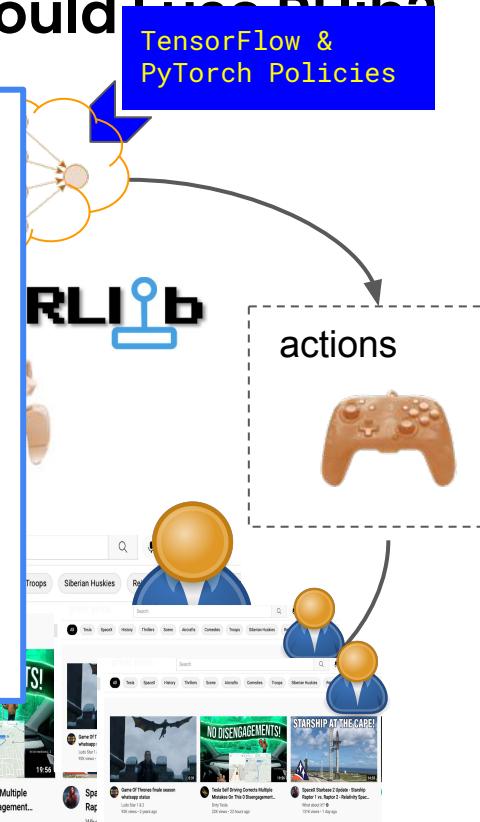


# And why should I use PyTorch?

```
# === Settings for Rollout Worker processes ===
# Number of rollout worker actors to create for parallel sampling. Setting
# this to 0 will force rollouts to be done in the trainer actor.
"num_workers": 2,
# Number of environments to evaluate vector-wise per worker. This enables
# model inference batching, which can improve performance for inference
# bottlenecked workloads.
"num_envs_per_worker": 1,
# If using num_envs_per_worker > 1, whether to create those new envs in
# remote processes instead of in the same worker. This adds overheads, but
# can make sense if your envs can take much time to step / reset
# (e.g., for StarCraft). Use this cautiously; overheads are significant.
"remote_worker_envs": False,
# Timeout that remote workers are waiting when polling environments.
# 0 (continue when at least one env is ready) is a reasonable default,
# but optimal value could be obtained by measuring your environment
# step / reset and model inference perf.
"remote_env_batch_wait_ms": 0,
```

```
def forward(self, input_dict, state, seq_lens):
    input_dict["obs"] = input_dict["obs"].float()
    fc_out, _ = self.torch_sub_model(input_dict, state, seq_lens)
    return fc_out, []

def value_function(self):
    return torch.reshape(self.torch_sub_model.value_function(), [-1])
```



## parallelize and distribute



```
# === Settings for Rollout Worker
# Number of rollout worker actors
# this to 0 will force rollouts to
"num_workers": 2,
# Number of environments to evaluate
# model inference batching, which
# bottlenecked workloads.
"num_envs_per_worker": 1,
# If using num_envs_per_worker > 1
# remote processes instead of in t
# can make sense if your envs can
# (e.g., for StarCraft). Use this
"remote_worker_envs": False,
# Timeout that remote workers are
# 0 (continue when at least one en
# but optimal value could be obtain
# step / reset and model inference
"remote_env_batch_wait_ms": 0,
```

## RLLib Algorithms

- High-throughput architectures
  - ⓘ ⚡ Distributed Prioritized Experience Replay (Ape-X)
  - ⓘ ⚡ Importance Weighted Actor-Learner Architecture (IMPALA)
  - ⓘ ⚡ Asynchronous Proximal Policy Optimization (APPO)
  - ⓘ ⚡ Decentralized Distributed Proximal Policy Optimization (DD-PPO)
- Gradient-based
  - ⓘ ⚡ Advantage Actor-Critic (A2C, A3C)
  - ⓘ ⚡ Deep Deterministic Policy Gradients (DDPG, TD3)
  - ⓘ ⚡ Deep Q Networks (DQN, Rainbow, Parametric DQN)
  - ⓘ ⚡ Policy Gradients
  - ⓘ ⚡ Proximal Policy Optimization (PPO)
  - ⓘ ⚡ Soft Actor Critic (SAC)
  - ⓘ ⚡ Slate Q-Learning (SlateQ)
- Derivative-free
  - ⓘ ⚡ Augmented Random Search (ARS)
  - ⓘ ⚡ Evolution Strategies
- Model-based / Meta-learning / Offline
  - ⓘ Single-Player AlphaZero (contrib/AlphaZero)
  - ⓘ ⚡ Model-Agnostic Meta-Learning (MAML)
  - ⓘ Model-Based Meta-Policy-Optimization (MBMPO)
  - ⓘ Dreamer (DREAMER)
  - ⓘ Conservative Q-Learning (CQL)
- Multi-agent
  - ⓘ QMIX Monotonic Value Factorisation (QMIX, VDN, IQN)
  - ⚡ Multi-Agent Deep Deterministic Policy Gradient (contrib/MADDPG)
- Offline
  - ⓘ ⚡ Advantage Re-Weighted Imitation Learning (MARWIL)
- Contextual bandits
  - ⓘ Linear Upper Confidence Bound (contrib/LinUCB)
  - ⓘ Linear Thompson Sampling (contrib/LinTS)
- Exploration-based plug-ins (can be combined with any algo)
  - ⓘ Curiosity (ICM: Intrinsic Curiosity Module)

25+ available algorithms  
(model-free/based; offline RL;  
meta RL; evolutionary strategies)



```
$ git clone https://github.com/sven1990/rllib_tutorials/production_rl_sun
$ cd rllib_tutorials/production_rl_sun
$ jupyter-lab
```



## RLib Algorithms

- High-throughput architecture
  - ⚡️ Distributed Priorit...

25+ available algorithms  
(model-free/based; offline RL;  
---> evolutionary strategies)

Tuned examples for most of our  
algorithms on popular environments.

master ray / rllib / tuned\_examples / ppo /

Go to file

Add file ▾

...



sven1977 [RLib] Slate-Q tf implementation and tests/benchmarks. (#22389)

✖ 6522935 17 days ago ⏲ History

..

atari-ddppo.yaml

[RLib] Deprecate vf\_share\_layers in top-level PPO/MAML/MB-MPO conf...

14 months ago

atari-ppo.yaml

[RLib] Deprecate vf\_share\_layers in top-

months ago

cartpole-appo-vtrace-fake-gpus.yaml

[RLib] Optionally don't drop last ts in v-trac

months ago

cartpole-appo-vtrace-separate-losses.yaml

[RLib] Add all simple learning tests as fram

months ago

cartpole-appo-vtrace.yaml

[RLib] Minor fixes/cleanups; chop\_into\_sequ

months ago

cartpole-appo.yaml

[RLib] Refactor: All tf static graph code sho

years ago

cartpole-ddppo.yaml

[RLib] Auto-framework, retire use\_pytorch

years ago

cartpole-grid-search-example.yaml

[RLib] Auto-framework, retire use\_pytorch

years ago

cartpole-ppo-fake-gpus.yaml

[RLib] Move existing fake multi-GPU learnin

years ago

cartpole-ppo-hyperband.yaml

[RLib] Auto-framework, retire use\_pytorch

years ago

cartpole-ppo.yaml

[RLib] Deprecate vf\_share\_layers in top-

months ago

frozenlake-appo-vtrace.yaml

[RLib] Upgrade gym version to 0.21 and dep

months ago

halfcheetah-appo.yaml

[RLib] Refactor: All tf static graph code sho

years ago

halfcheetah-ppo.yaml

[RLib] Auto-framework, retire use\_pytorch

years ago

hopper-ppo.yaml

[RLib] Auto-framework, retire use\_pytorch

years ago

humanoid-ppo-gae.yaml

[RLib] Auto-framework, retire use\_pytorch in favor of `framework=...

2 years ago

20 lines (20 sloc) | 506 Bytes

```
1 cartpole-appo:  
2   env: CartPole-v0  
3   run: APPO  
4   stop:  
5     episode_reward_mean: 150  
6     timesteps_total: 200000  
7   config:  
8     # Works for both torch and tf.  
9     framework: tf  
10    num_envs_per_worker: 5  
11    num_workers: 1  
12    num_gpus: 0  
13    observation_filter: MeanStdFilter  
14    num_sgd_iter: 6  
15    vf_loss_coeff: 0.01  
16    vtrace: false  
17    model:  
18      fcnet_hiddens: [32]  
19      fcnet_activation: linear  
20      vf_share_layers: true
```

ale

```
$ git clone https://git  
$ cd rllib_tutorials/p  
$ jupyter-lab
```



# And why should I use RLlib?

Because these companies here do!

Thx for presenting

Summits!



TWO SIGMA

```
$ git clone https://github.com/ray-project/rllib_tutorials.git  
$ cd rllib_tutorials/pyspark  
$ jupyter-lab
```

Tuned examples for most of our algorithms on popular environments.

master ray / rllib / tuned\_examples / ppo /

sven1977 [RLlib] Slate-Q tf implementation and tests/benchmarks. (#22389) 6522935 17 days ago History

..

atari-ddppo.yaml [RLlib] Deprecate vf\_share\_layers in top-level PPO/MAML/MB-MPO conf... 14 months ago

atari-ppo.yaml [RLlib] Deprecate vf\_share\_layers in top... 14 months ago

cartpole-appo-vtrace-fake-gpus.yaml [RLlib] Optionally don't drop last ts in v-trac... 14 months ago

cartpole-appo-vtrace-separate-losses.yaml [RLlib] Add all simple learning tests as fram... 14 months ago

cartpole-appo-vtrace.yaml [RLlib] Minor fixes/cleanups; chop\_into\_sequenc... 14 months ago

cartpole-appo.yaml [RLlib] Refactor: All tf static graph code sho... 14 months ago

cartpole-ddppo.yaml [RLlib] Auto-framework, retire use\_pytorch... 14 months ago

cartpole-grid-search-example.yaml [RLlib] Auto-framework, retire use\_pytorch... 14 months ago

cartpole-ppo-fake-gpus.yaml [RLlib] Move existing fake multi-GPU learnin... 14 months ago

cartpole-ppo-hyperband.yaml [RLlib] Auto-framework, retire use\_pytorch... 14 months ago

cartpole-ppo.yaml [RLlib] Deprecate vf\_share\_layers in top... 14 months ago

frozenlake-appo-vtrace.yaml [RLlib] Upgrade gym version to 0.21 and dep... 14 months ago

halfcheetah-appo.yaml [RLlib] Refactor: All tf static graph code sho... 14 months ago

halfcheetah-ppo.yaml [RLlib] Auto-framework, retire use\_pytorch... 14 months ago

hopper-ppo.yaml [RLlib] Auto-framework, retire use\_pytorch... 14 months ago

humanoid-ppo-gae.yaml [RLlib] Auto-framework, retire use\_pytorch in favor of `framework=... 2 years ago

20 lines (20 sloc) | 506 Bytes

```
1 cartpole-appo:  
2   env: CartPole-v0  
3   run: APPO  
4   stop:  
5     episode_reward_mean: 150  
6     timesteps_total: 200000  
7   config:  
8     # Works for both torch and tf.  
9     framework: tf  
10    num_envs_per_worker: 5  
11    num_workers: 1  
12    num_gpus: 0  
13    observation_filter: MeanStdFilter  
14    num_sgd_iter: 6  
15    vf_loss_coeff: 0.01  
16    vtrace: false  
17    model:  
18      fcnet_hiddens: [32]  
19      fcnet_activation: linear  
20      vf_share_layers: true
```

ale

# 15 min Break :)

# Then ... moving to our Jupyter Notebook



The screenshot shows a Jupyter Notebook interface running in a web browser. On the left, a file browser window displays a directory named 'Images' containing various PNG files related to reinforcement learning and recommendation systems. On the right, a notebook cell titled 'Reinforcement Learning for Recommender Systems' is open, showing code and several small images illustrating different RL concepts like bandits, slate-Q, and RLlib.

**Reinforcement Learning for Recommender Systems**

From Contextual Bandits to Slate-Q

Overview

"Industry-grade, hands-on RL with Ray RLlib" is a tutorial for industry researchers, domain-experts, and ML-engineers, showcasing ...

1... how you can use RLlib to build a recommender system simulator for your industry applications and run a slate-capable algorithm against this simulator.  
2... how RLlib's offline algorithms pose solutions in case you don't have a simulator of your problem environment at hand.

We will further explore how to deploy one or more trained models to production using Ray Serve and how RLlib's bandit algorithms could be used to select the best model from some set of candidates for that purpose.

During the live-coding phases, we will build a recommender system simulating environment with RLlib and google's RecSim, choose, configure, and run an RLlib algorithm, and experiment and tune hyperparameters with Ray Tune.

RLlib offers industry-grade scalability, a large list of algos to choose from (offline, model-based, model-free, etc.), support for TensorFlow and PyTorch, and a unified API for a variety of applications. This tutorial includes a brief introduction to provide an overview of concepts (e.g. why RL?) before proceeding to RLlib (recommender systems) environments, neural network models, offline RL, student exercises, Q&A, and more. All code will be provided as .py files in a GitHub repo.



# Google RecSim

## building our own recomm. sys. simulator

<https://github.com/google-research/recsim>

```
$ pip install recsim
```

**RecSim comes with 3 built-in example environment that can be used as-is by RLlib:**

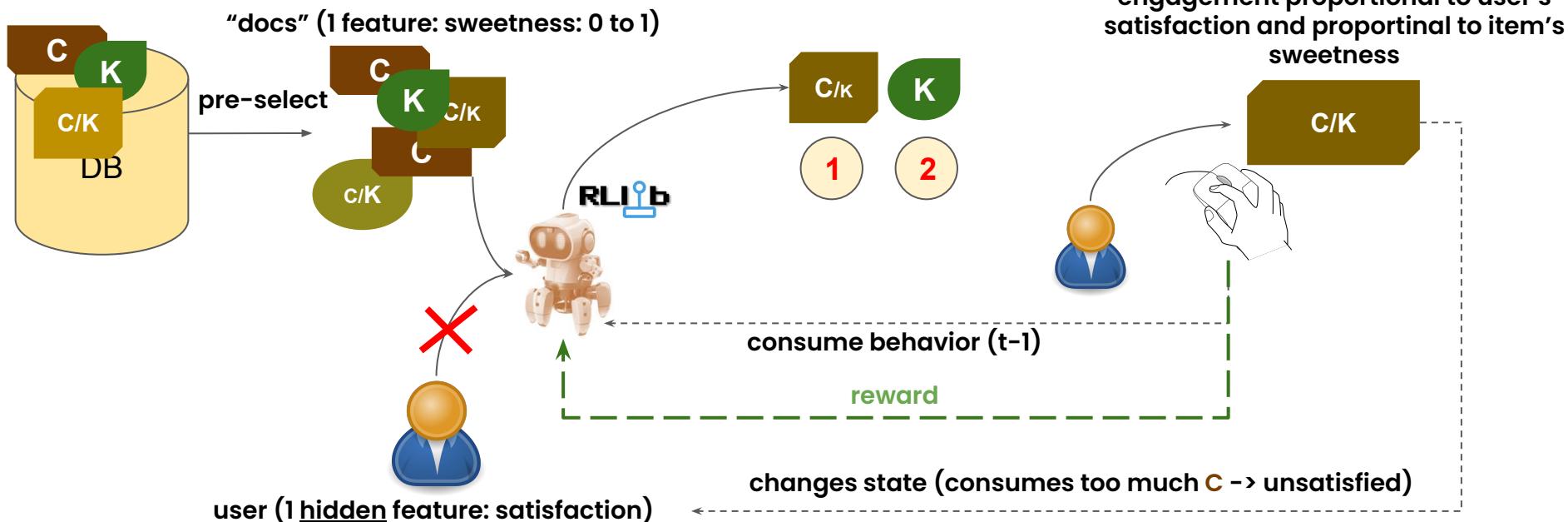
- **Long Term Satisfaction**
- **Interest Evolution**
- **Interest Exploration**

**K**  
Kale: Sweetness=0.0  
Low engagement; high satisfaction

**C**  
Chocolate: Sweetness=1.0  
High engagement; low satisfaction

# Google RecSim

## The “Long Term Satisfaction” Problem



<https://github.com/google-research/recsim>

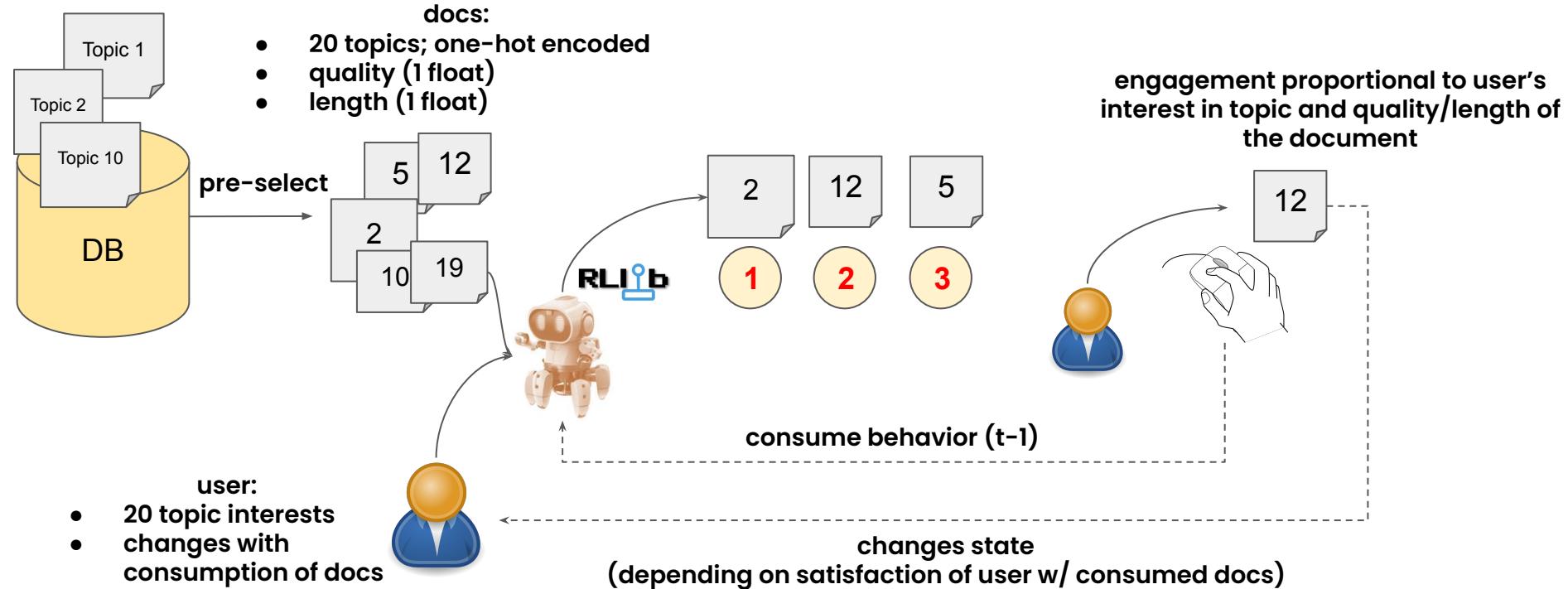
\$ pip install recsim

anyscale



# Google RecSim

## The “Interest Evolution” Problem



<https://github.com/google-research/recsim>

\$ pip install recsim



# Contextual n-armed Bandit



$$A_t = \arg \max_a \left[ Q_t(a) + c \sqrt{\frac{\log t}{N_t(a)}} \right]$$

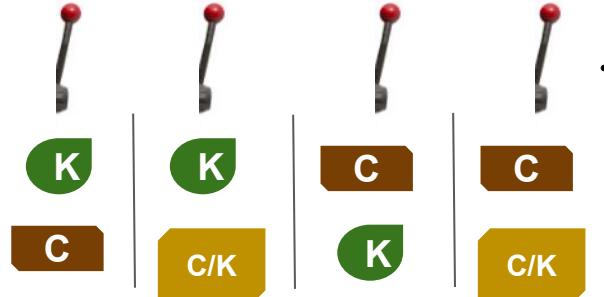
exploration/exploitation

"context" (observation)

n candidate docs (vectors)

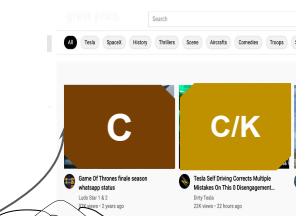
pick 1 arm (action)

arms (flattened action space)



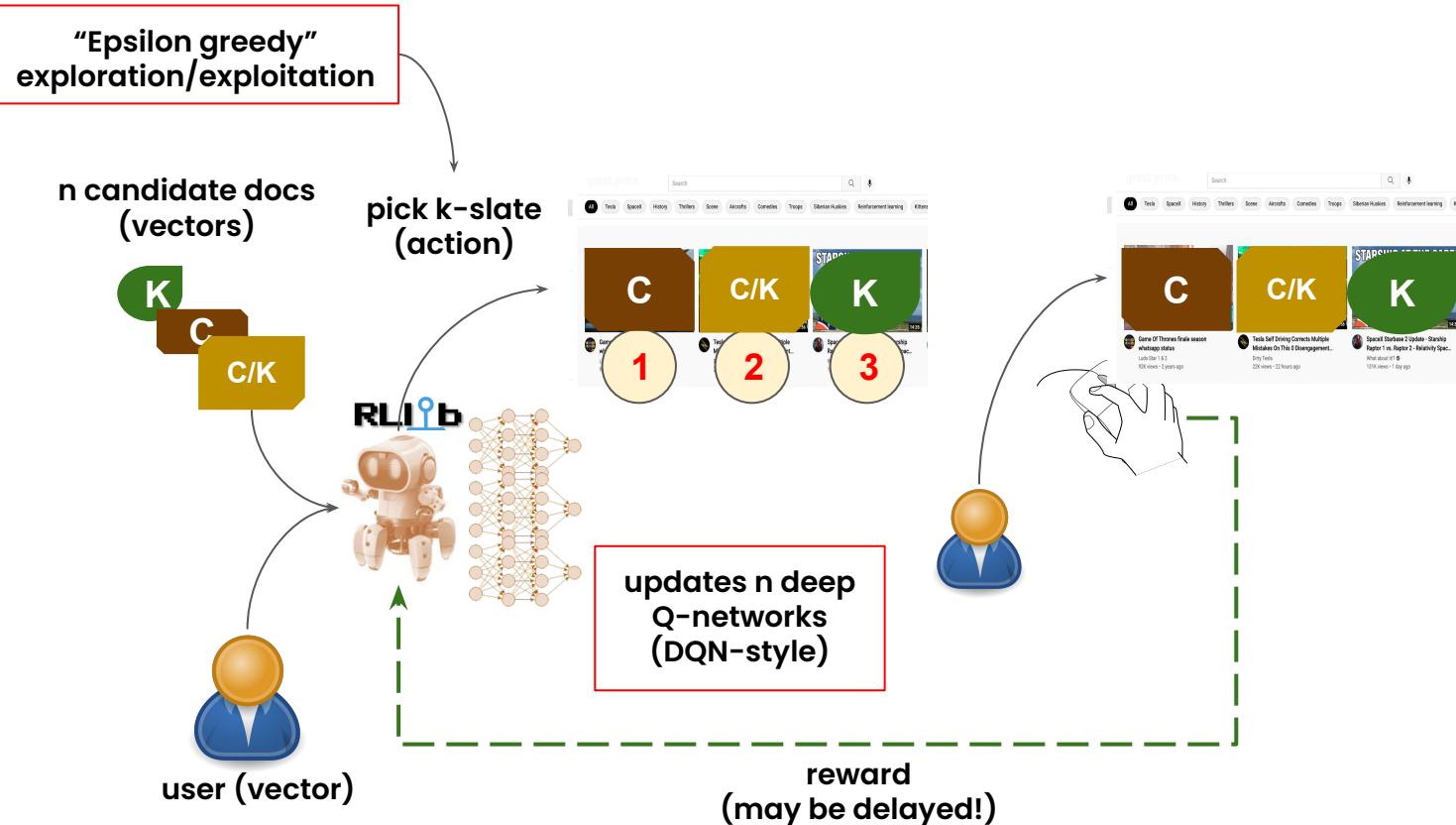
updates linear reward model (after each timestep; "online")

reward  
(must be immediate)





# Slate-Q Algorithm

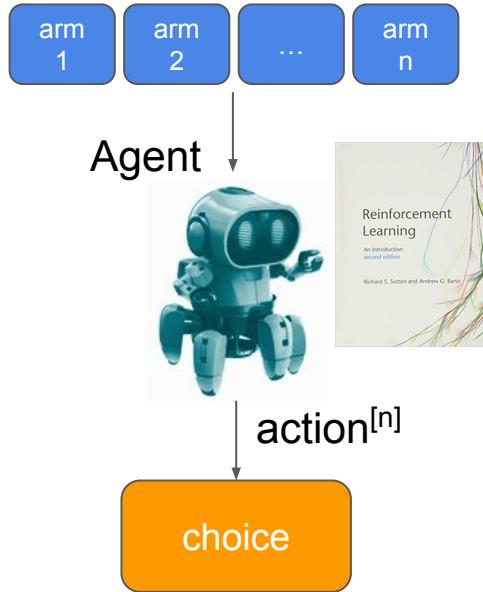


# Which Algorithms are we going to use ... and how do they work?



**Bandit**

**One-armed  
Bandit**



**$n$ -armed  
Bandit**

# Simple, n-Armed Bandit



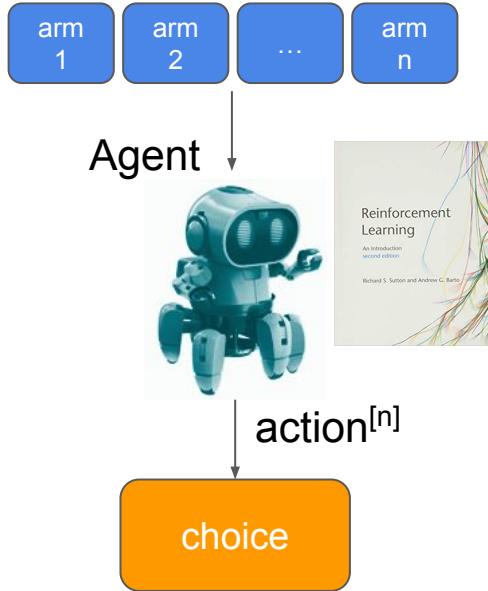
**n-armed  
Bandit**

- Models each arm's reward signal ("Q-function").
- Needs to carefully balance exploration (i.e. pick an arm that currently has a high Q-estimate, but also still lots of uncertainty (optimism in the face of uncertainty), e.g. UCB sampling:

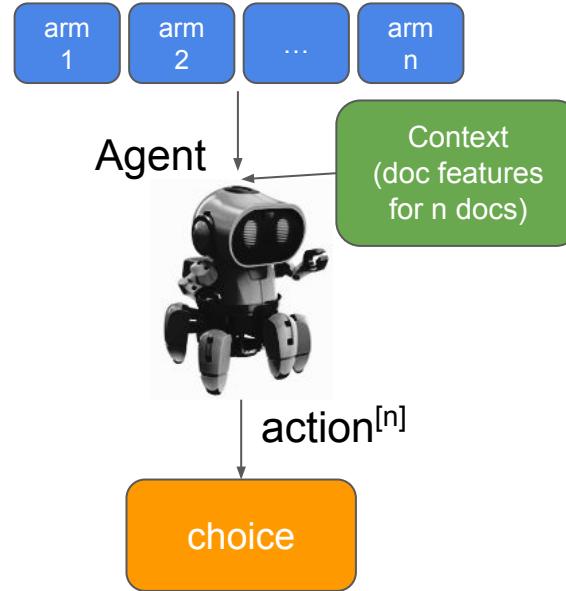
$$A_t = \arg \max_a \left[ Q_t(a) + c \sqrt{\frac{\log t}{N_t(a)}} \right]$$

- Received rewards are used to update Q-function estimates for the picked arm.
- No "context"; Each arm always uses same underlying (fixed) model.

# Contextual Bandit

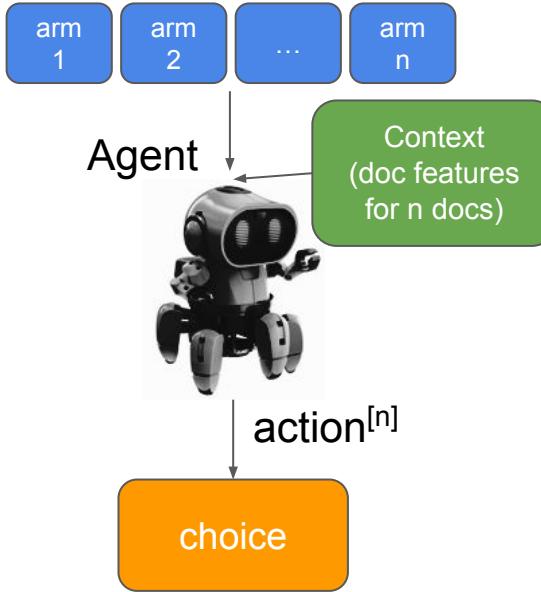


**n-armed  
Bandit**



**contextual  
n-armed  
Bandit**

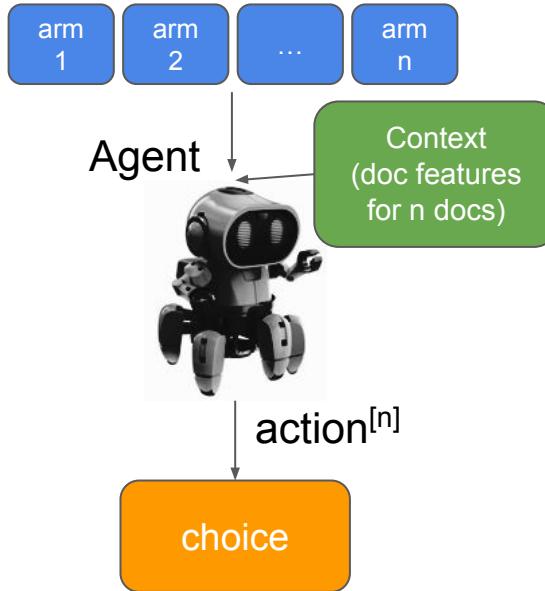
# Contextual Bandit



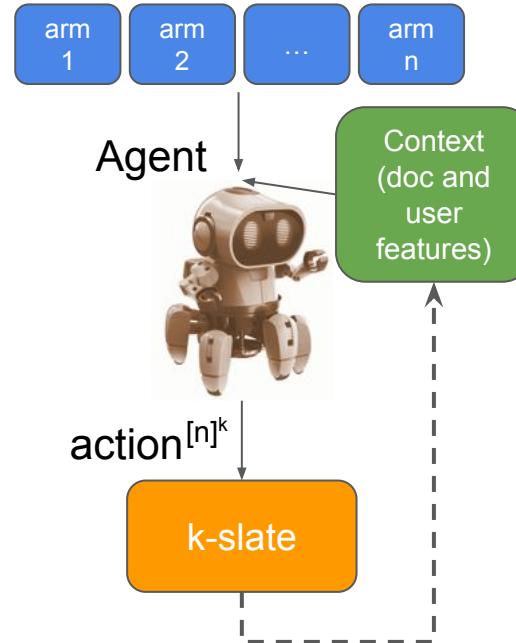
**contextual  
n-armed  
Bandit**

- Has “context”: Each arm represents one document choice and is characterized by the respective doc’s feature vector.
- Models one user and her behavior towards the different presented docs (will pick & consume?); E.g. learns a covariance matrix that can be multiplied by the different doc vectors to yield estimated rewards.
- Exploration: Same as simple n-armed bandit.
- Updates: Same as simple, n-armed bandit.
- Can emulate k-slate actions by “flattening” MultiDiscrete action space into Discrete one.

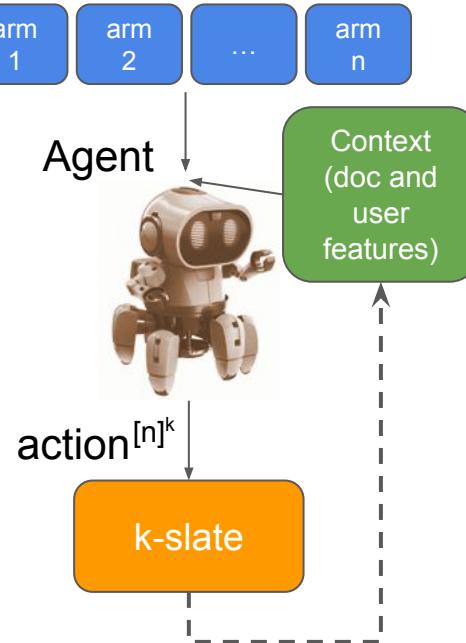
# Slate-Q



**contextual  
 $n$ -armed  
Bandit**



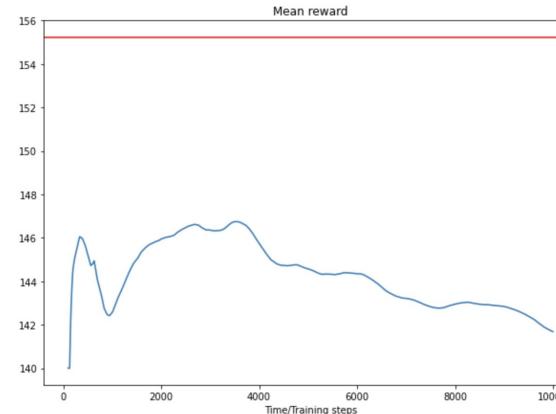
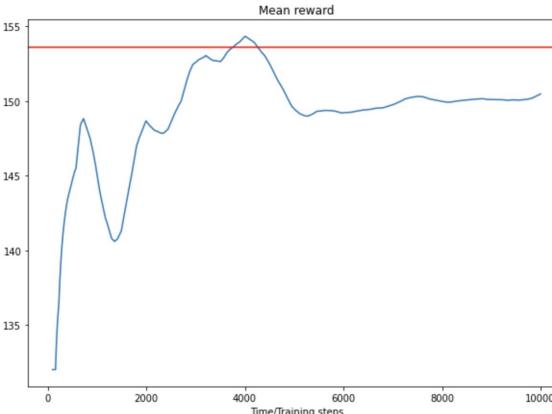
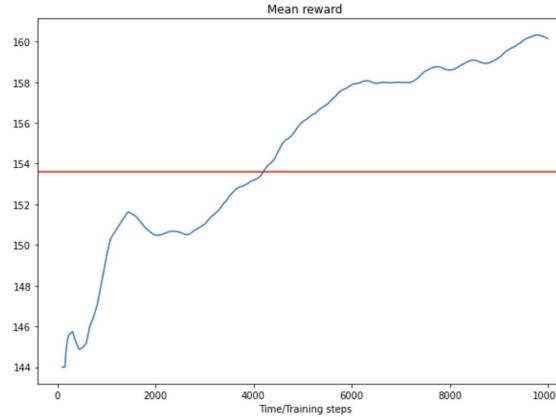
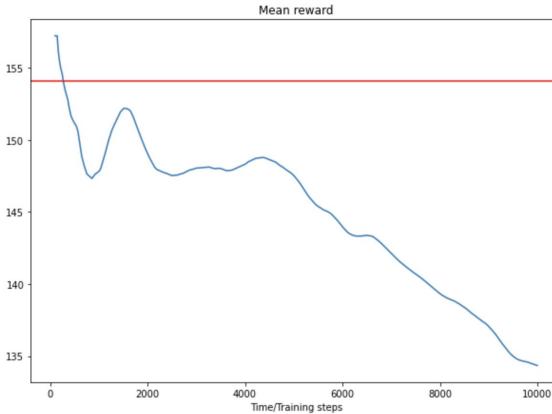
**Slate-Q**



- Sees both “doc” and “user” information and can thus learn to model different users.
- Policy is a set of  $n$  Q-heads, where  $n$  is the number of document candidates to pick from. Input to each of these  $n$  heads is the user vector + the respective doc vector.
- Exploration: Epsilon-greedy.
- Updates: Analogous to DQN, but using slate-Q-value decomposition.
- Supports k-slates, multiple users, and longer user sessions (episodes that are longer than 1 timestep).

# Contextual UCB Bandit Results

10k ts; InterestEvolution; 50 candidates; slate-size=2



# And now ... Moving to our Jupyter Notebook



The screenshot shows a Jupyter Notebook environment with the following details:

- File Explorer:** On the left, a file browser titled "tutorial\_notebook.ipynb" shows a directory structure under "/images/". The files listed include: rl\_cycle\_and\_slateq.png, contextual\_bandit.png, dcos2.jpg, environment.png, forklifts.jpg, offline\_rl.png, ptfar.jpg, rl\_cycle.png, rl\_algorithms.png, rl\_band\_tune.png, robots.jpg, slateq.png, spaces.png, spotify.jpg, tests.jpg, unity3d\_blog\_post.png, and youtube.png.
- Notebook Content:** The main area displays a notebook cell with the title "Reinforcement Learning for Recommender Systems" and subtitle "From Contextual Bandits to Slate-Q". Below the titles are several small images illustrating different RL concepts.
- Overview:** A section titled "Overview" provides a brief introduction:

"Industry-grade, hands-on RL with Ray RLlib" is a tutorial for industry researchers, domain-experts, and ML-engineers, showcasing ...

1... how you can use RLlib to build a recommender system simulator for your industry applications and run a slate-capable algorithm against this simulator.  
2... how RLlib's offline algorithms pose solutions in case you don't have a simulator of your problem environment at hand.

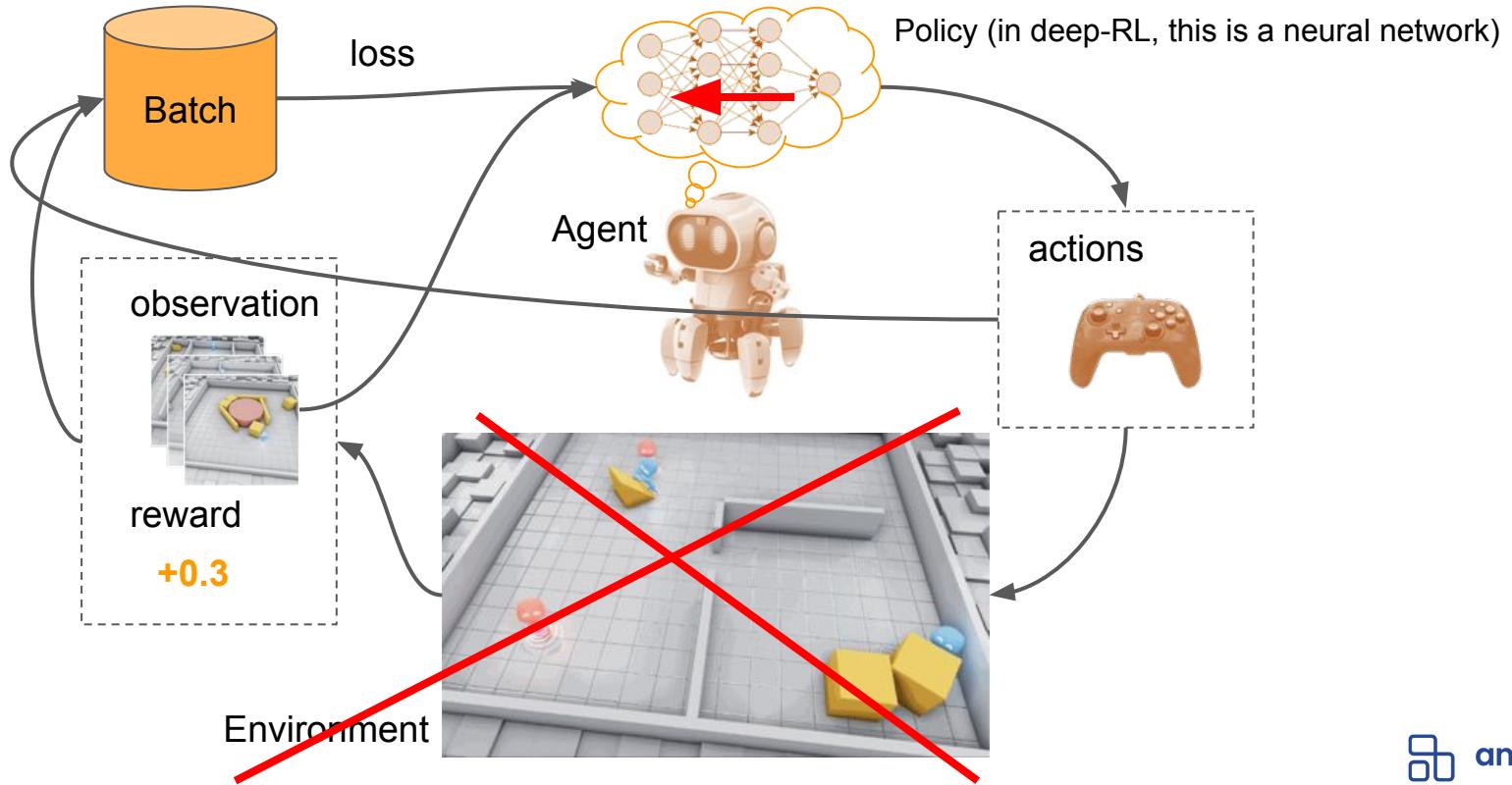
We will further explore how to deploy one or more trained models to production using Ray Serve and how RLlib's bandit algorithms could be used to select the best model from some set of candidates for that purpose.

During the live-coding phases, we will build a recommender system simulating environment with RLlib and google's RecSim, choose, configure, and run an RLlib algorithm, and experiment and tune hyperparameters with Ray Tune.

RLlib offers industry-grade scalability, a large list of algos to choose from (offline, model-based, model-free, etc.), support for TensorFlow and PyTorch, and a unified API for a variety of applications. This tutorial includes a brief introduction to provide an overview of concepts (e.g. why RL?) before proceeding to RLlib (recommender systems) environments, neural network models, offline RL, student exercises, Q&A, and more. All code will be provided as .py files in a GitHub repo.

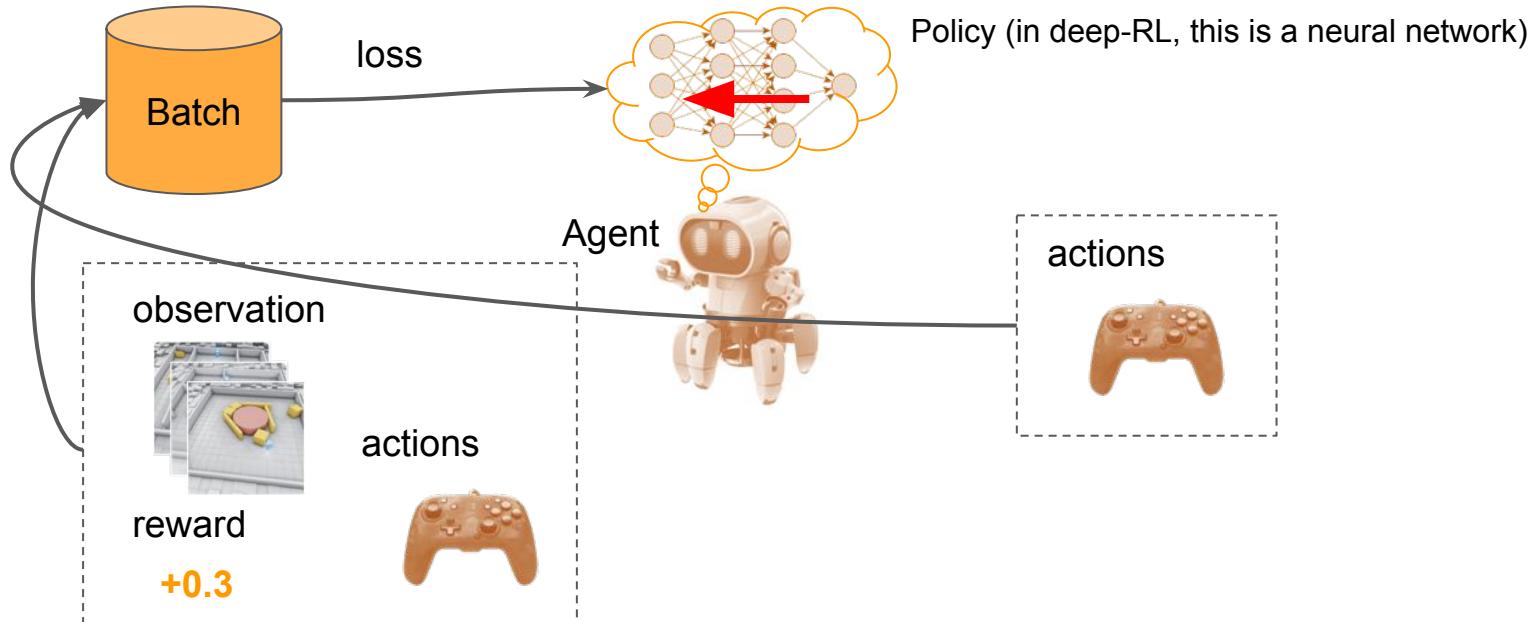
# What's Offline Reinforcement Learning?

Aka: "Batch RL"



# What's Offline Reinforcement Learning?

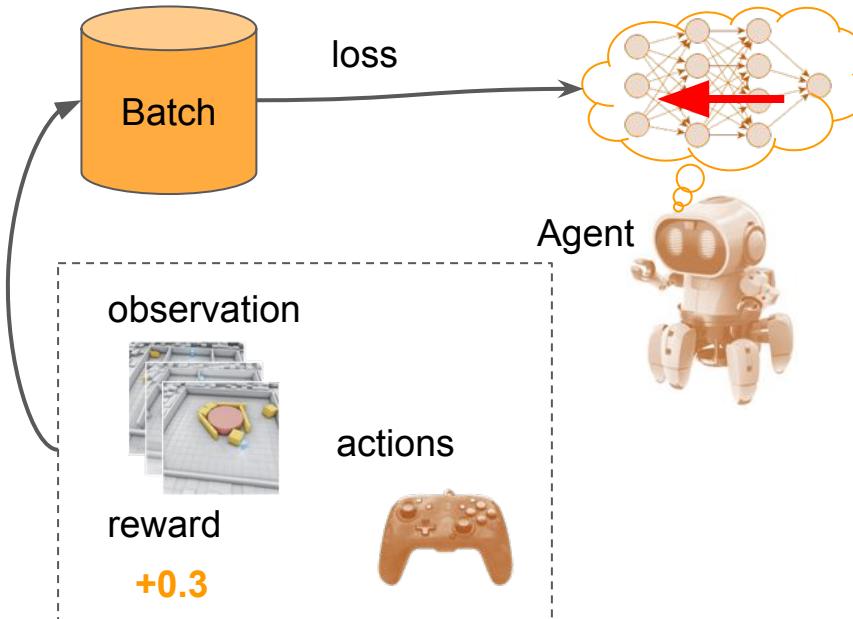
Aka: Batch RL



Historic data recorded in the past (e.g. a JSON file!).

# What's Offline Reinforcement Learning?

Aka: Batch RL



Policy (in deep-RL, this is a neural network)

## 2 ways of doing this:

- Behavioral Cloning (BC), aka: “Imitation learning”:  $\text{loss} = -\log(p(a_h))$ 
  - Don’t care about rewards.
  - Pure SL: Observations=inputs; actions=labels
- Offline RL: Don’t only imitate the historic policy, but also try to improve over it.
  - Rewards are needed for computing losses.

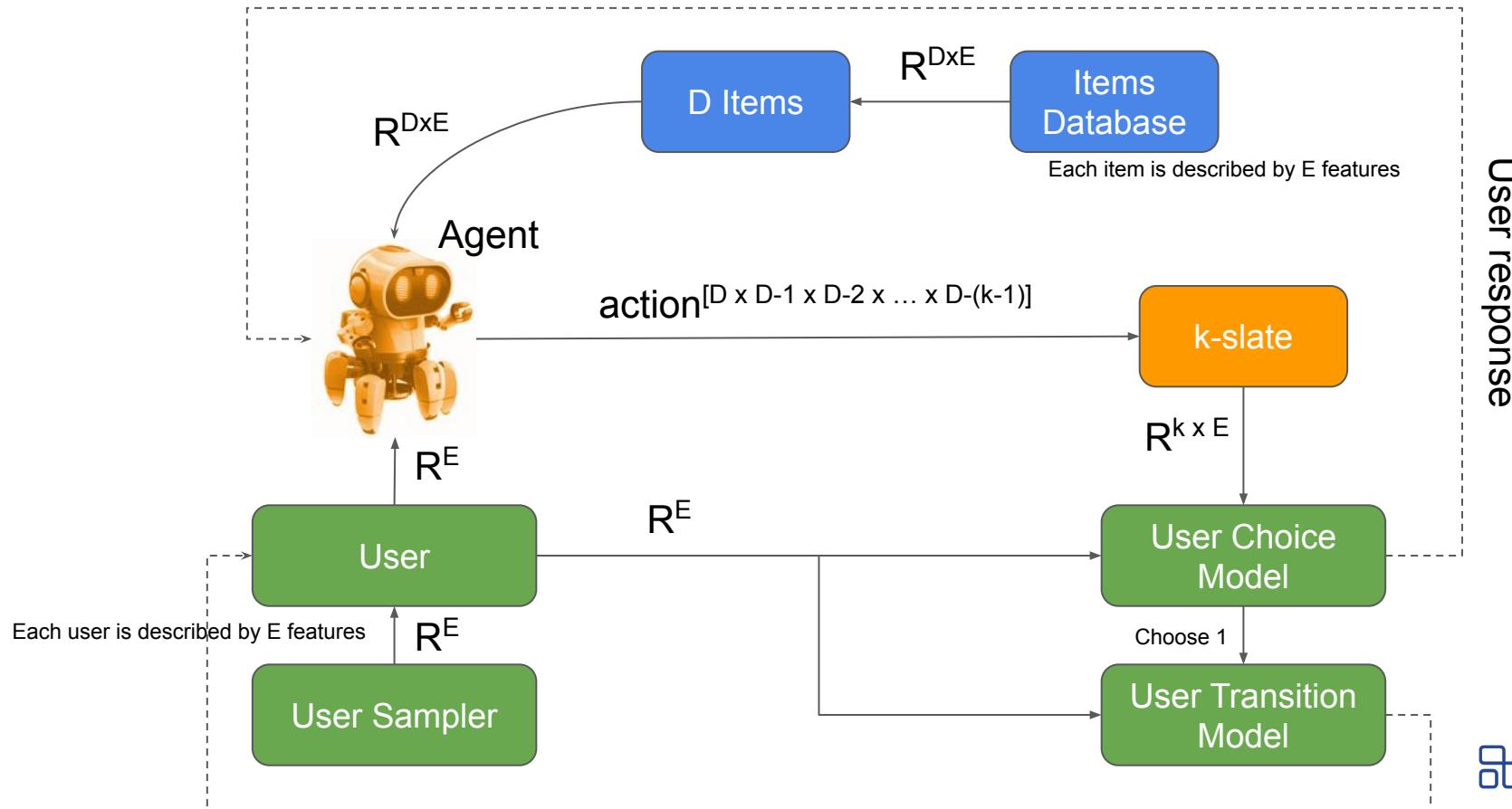
Historic data recorded in the past (e.g. a JSON file!).

# And now ... Moving to our Jupyter Notebook



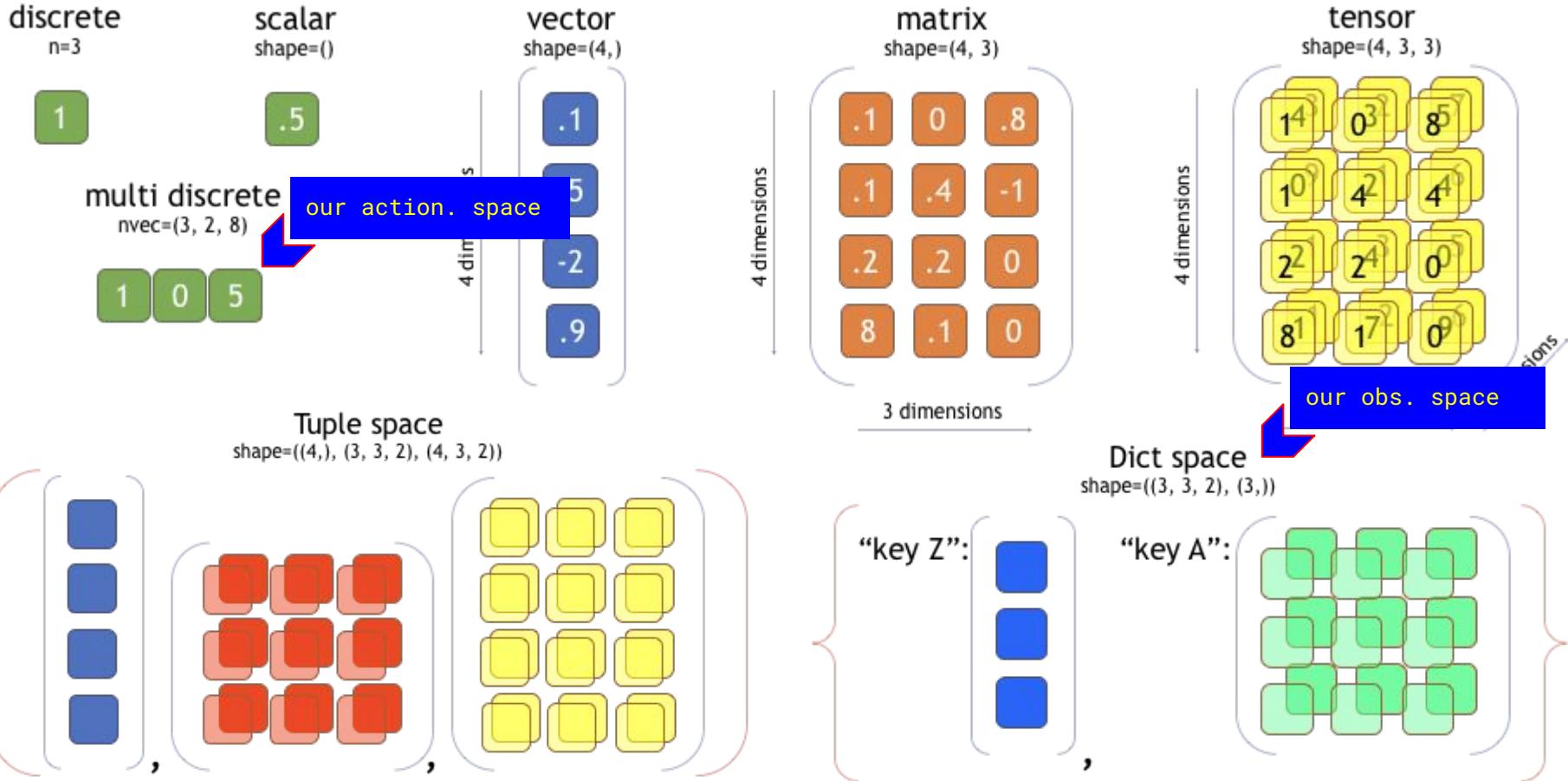
The screenshot shows a Jupyter Notebook environment with the following details:

- File Explorer:** On the left, a file browser titled "tutorial\_notebook.ipynb" shows a directory structure under "/images/". The files listed include: rl\_cycle\_and\_slateq.png, contextual\_bandit.png, dcos2.jpg, environment.png, forklifts.jpg, offline\_rl.png, ptfar.jpg, rl\_cycle.png, rl\_algorithms.png, rl\_band\_tune.png, robots.jpg, slateq.png, spaces.png, spotify.jpg, tests.jpg, unity3d\_blog\_post.png, and youtube.png.
- Notebook Content:** The main area displays a notebook cell with the title "Reinforcement Learning for Recommender Systems" and subtitle "From Contextual Bandits to Slate-Q". Below the titles are several small images illustrating different RL concepts.
- Overview:** A section titled "Overview" provides a brief introduction to the tutorial, mentioning "Industry-grade, hands-on RL, with Ray RLlib" and its purpose of showcasing RLlib's capabilities.
- Text:** The text continues with instructions on how to use RLlib to build a recommender system simulator and how RLlib's offline algorithms pose solutions in case you don't have a simulator of your problem environment at hand.
- Conclusion:** It ends with a note about deploying trained models to production using Ray Serve and tuning RLlib's bandit algorithms.
- Footnote:** A small note at the bottom right discusses RLlib's scalability and its support for TensorFlow and PyTorch.





# What's a Space?



# Quick demo of notebook