Reinforcement Learning for Recommender Systems

From Contextual Bandits to Slate-Q

Sven Mika, PhD
https://linkedin.com/in/sven-mika
sven@anyscale.com

anyscale

RLlib

Who is the RL Team @ Anyscale?

https://docs.ray.io

Sven

Jun

Avnish

Artur

Steven

Christy (Developer Advocate)

Summer ‘22:

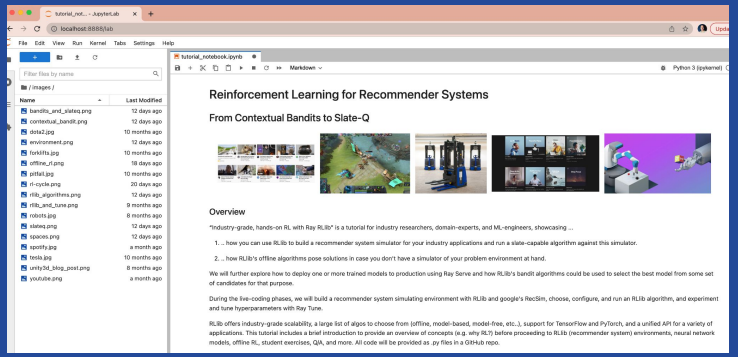
Kourosh

Justin

anyscale

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

anyscale

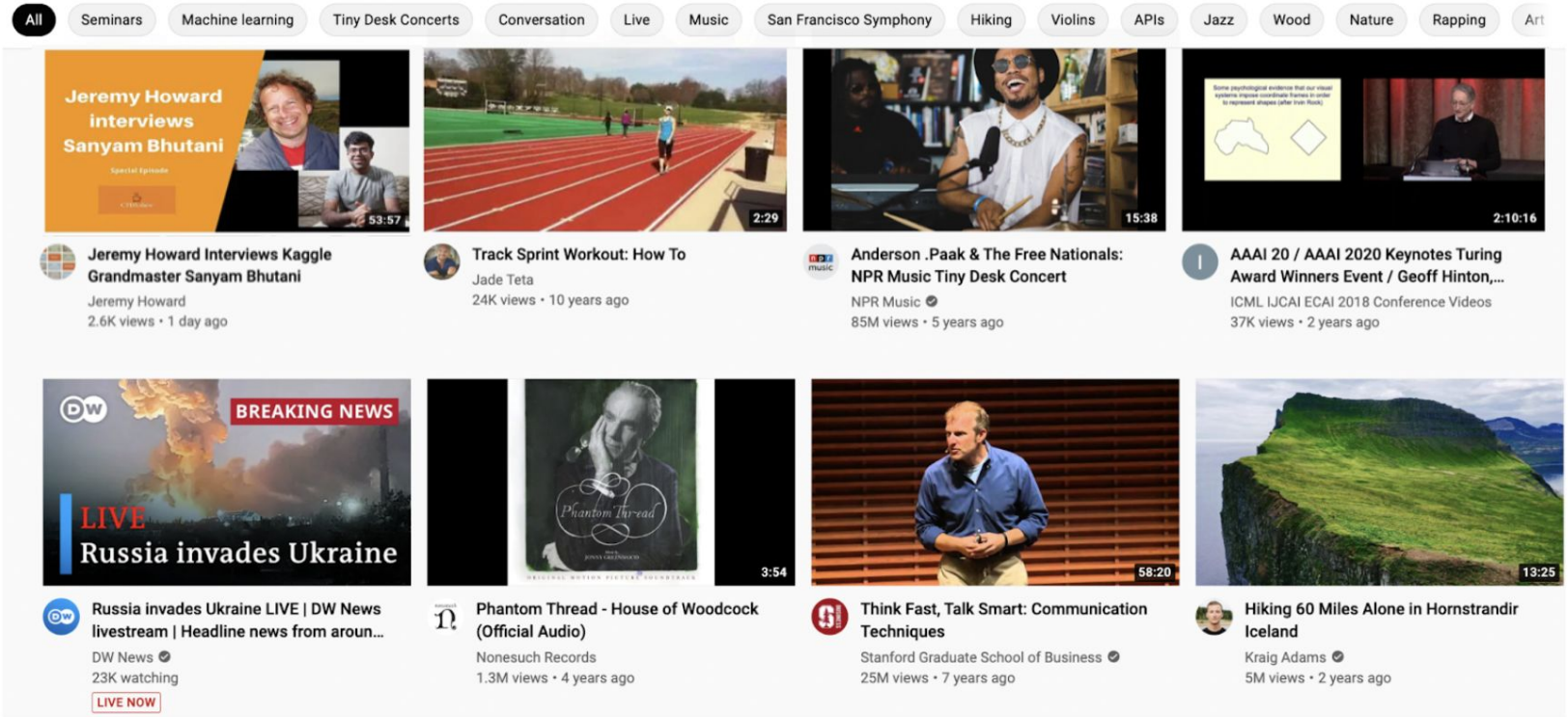
Overview of the tutorial

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

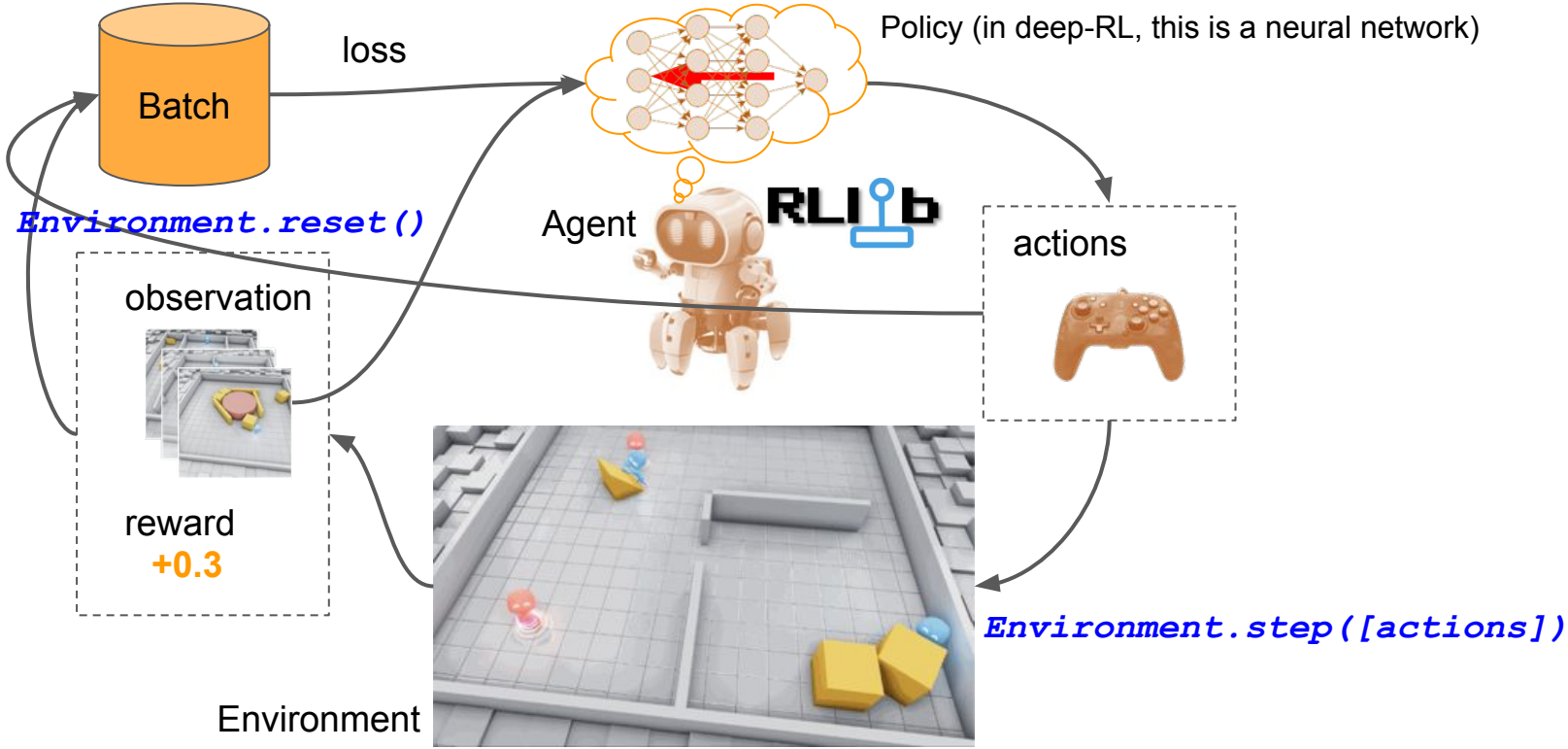
$ git clone https://github.com/sven1977/rllib_tutorials
$ cd rllib_tutorials/production_rl_summit_2022
$ jupyter-lab

anyscale

Recommender Systems
(Personalization Systems)



What's Reinforcement Learning (RL)?

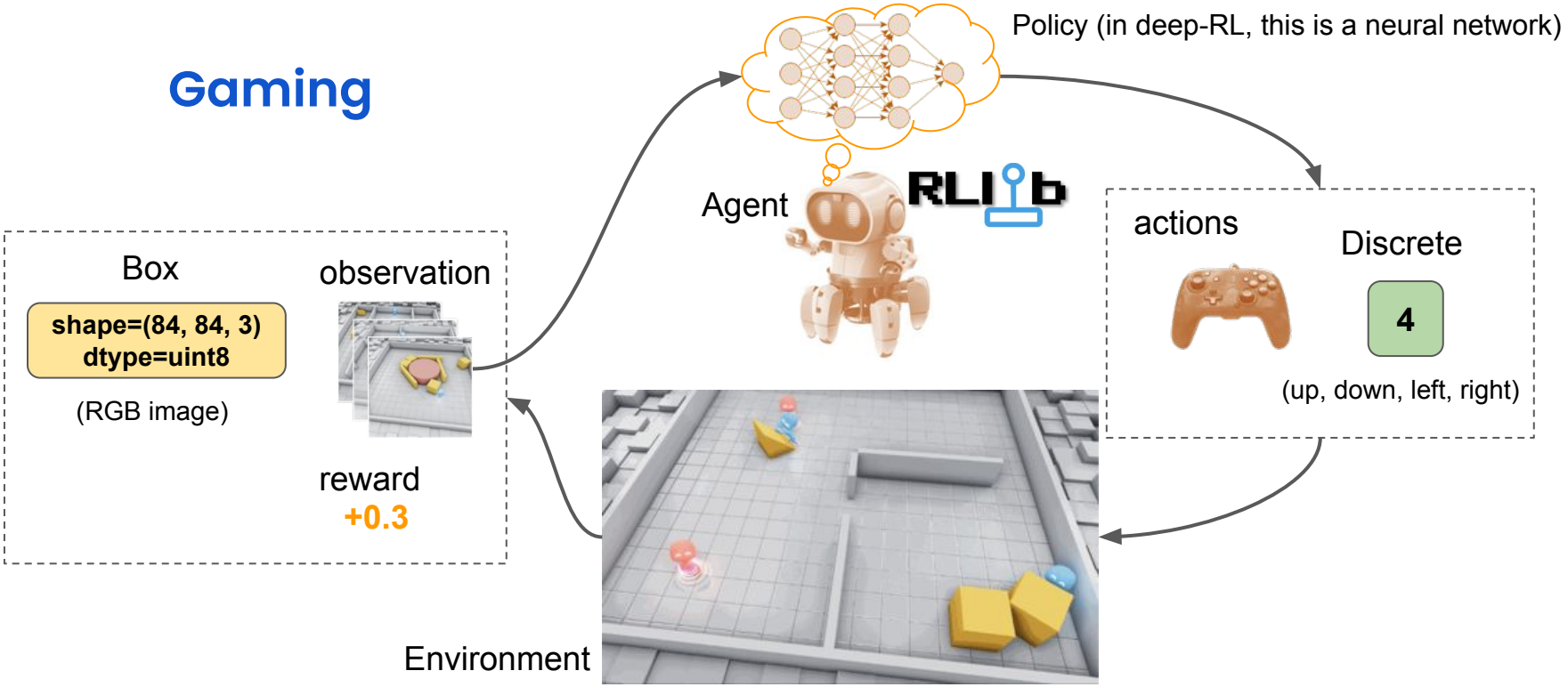


$ git clone https://github.com/sven1977/rllib_tutorials
$ cd rllib_tutorials/production_rl_summit_2022
$ jupyter-lab

anyscale

What's Reinforcement Learning (RL)?

Gaming

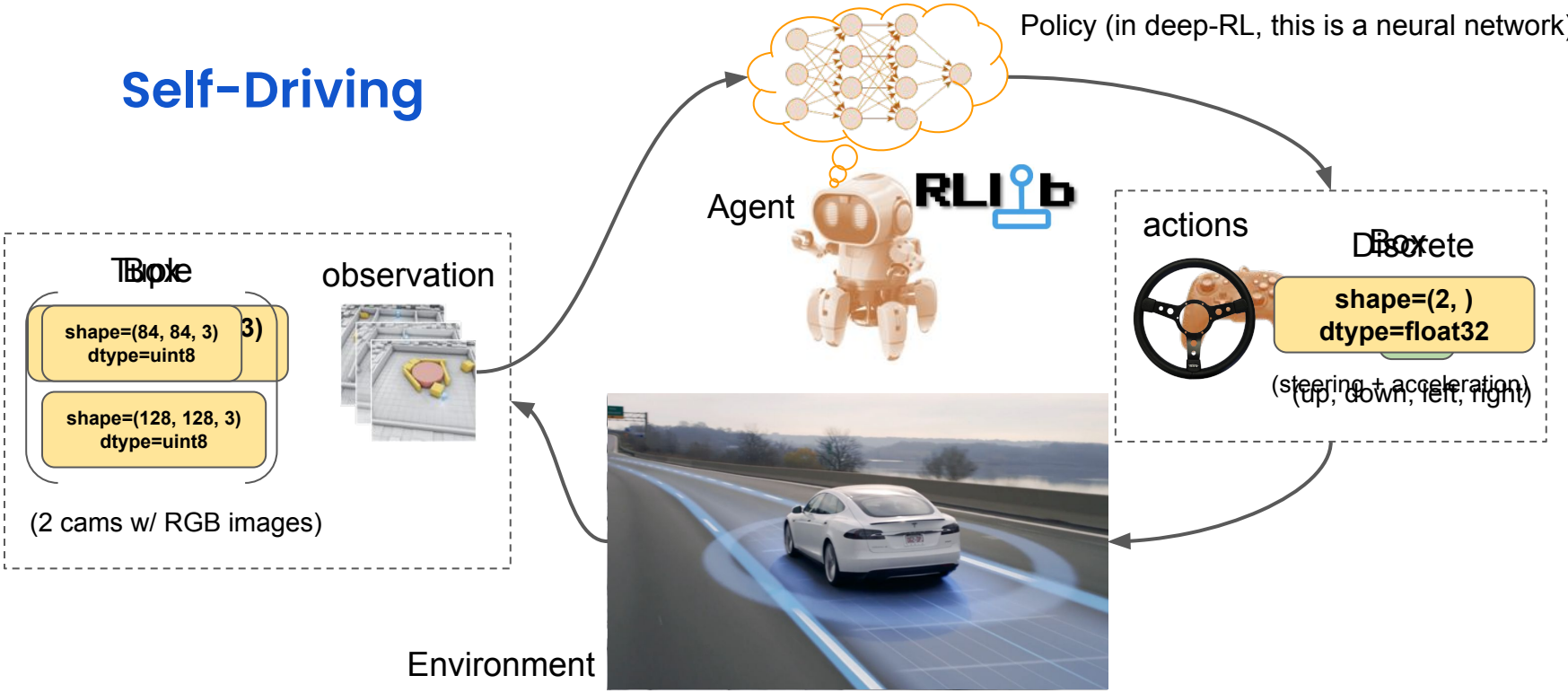


$ git clone https://github.com/sven1977/rllib_tutorials
$ cd rllib_tutorials/production_rl_summit_2022
$ jupyter-lab

anyscale

What's Reinforcement Learning (RL)?

Self-Driving



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

anyscale

What's Reinforcement Learning (RL)?

Recommender Systems

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

Agent

actions

MultiDiscrete

Box

100 100 100

1 2 3

(k-slate out of 100 candidates)

Dict

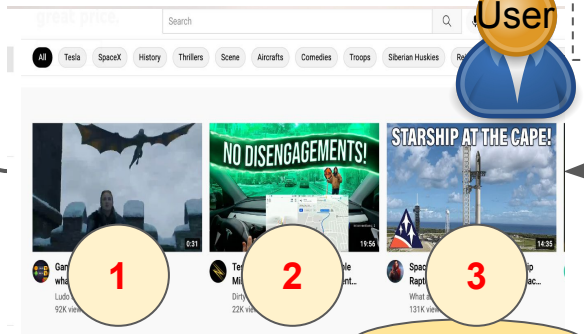
observation

doc: shape=(100, E) dtype=float

user: shape=(E,) dtype=float

(2 cams w/ RGB images)

User



Environment

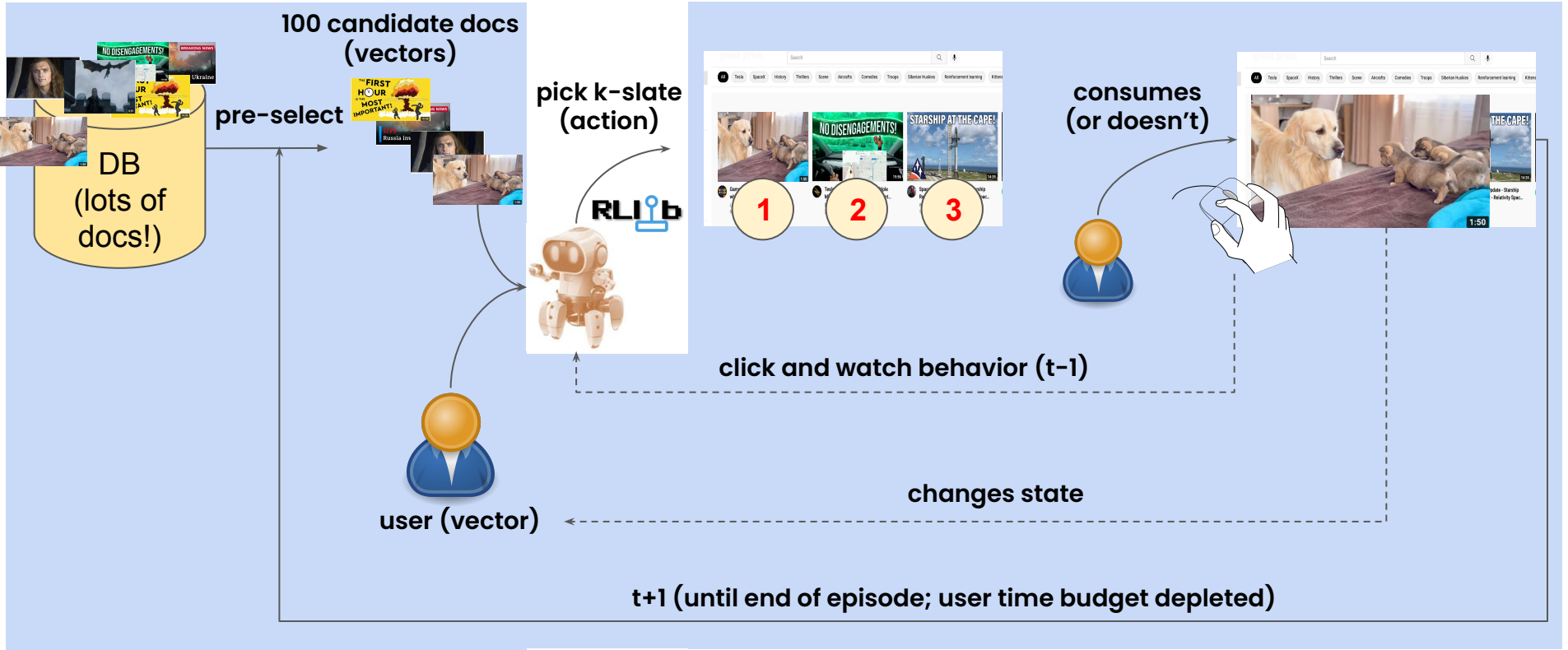
x 100

100 x 99 x 98 = ~1M unique slates

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

anyscale

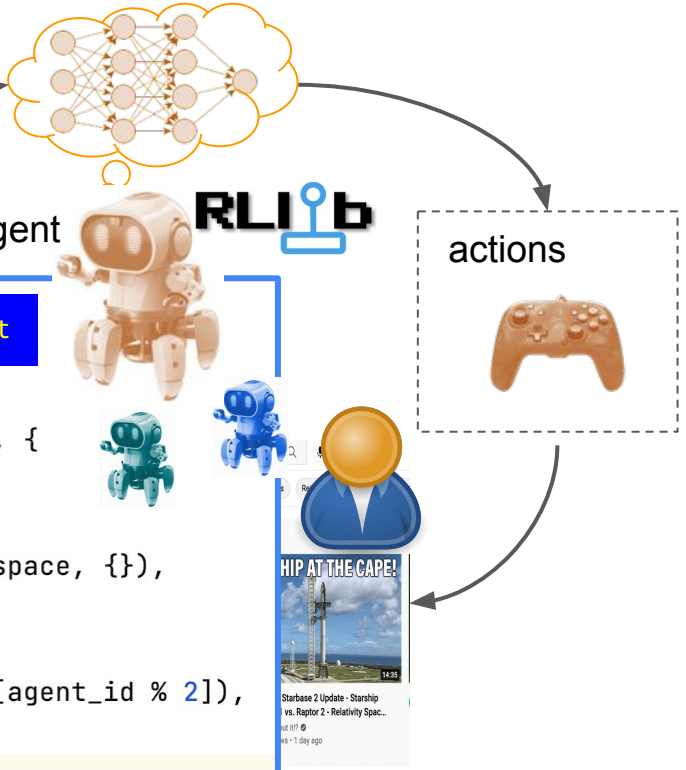
A Recommender System in Action



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

anyscale

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

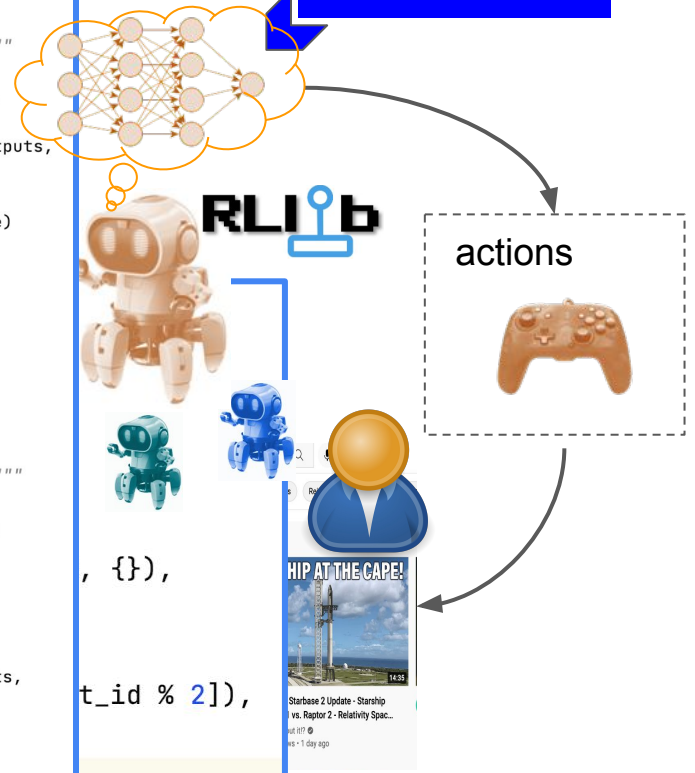
anyscale

And why should Ray RLlib?

```
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])
```



anyscale

And why should

```
    # === 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,
```

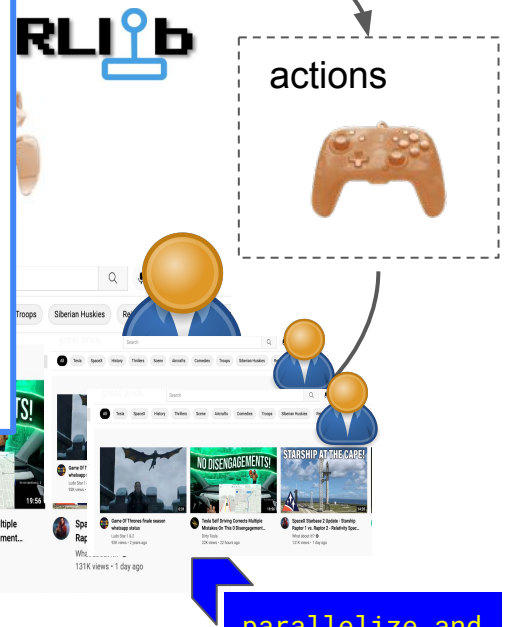
```
    },
}
```

```
ace, num_outputs,
```

```
    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])
```

```
$ git cl
$ cd rl
$ jupyt
```

TensorFlow & PyTorch Policies



parallelize and distribute

anyscale

RLlib

```
# === Settings for Rollout Worker
# Number of rollout worker actors
# this to 0 will force rollouts to
"num_workers": 2,
# Number of environments to evalua
# 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 obtai
# 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)

llelize and
ribute

$ git clone https://github.com/sven19
$ cd rllib_tutorials/production_rl_sun
$ jupyter-lab

anyscale

RLlib

RLlib Algorithms

- High-throughput architectures
  - Distributed Priorit

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

Tuned examples for most of our algorithms on popular environments.

master ▾ | ray / rllib / tuned_examples / ppo / | Go to file | Add file ▾ | ⋯

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-	nths ago
cartpole-appo-vtrace-fake-gpus.yaml	[RLlib] Optionally don't drop last ts in v-trac	nths ago
cartpole-appo-vtrace-separate-losses.yaml	[RLlib] Add all simple learning tests as `fram`	nths ago
cartpole-appo-vtrace.yaml	[RLlib] Minor fixes/cleanups; chop_into_seq	nths ago
cartpole-appo.yaml	[RLlib] Refactor: All tf static graph code sho	nths ago
cartpole-ddppo.yaml	[RLlib] Auto-framework, retire `use_pytorch`	ears ago
cartpole-grid-search-example.yaml	[RLlib] Auto-framework, retire `use_pytorch`	ears ago
cartpole-ppo-fake-gpus.yaml	[RLlib] Move existing fake multi-GPU learnin	nths ago
cartpole-ppo-hyperband.yaml	[RLlib] Auto-framework, retire `use_pytorch`	ears ago
cartpole-ppo.yaml	[RLlib] Deprecate `vf_share_layers` in top-	nths ago
frozenlake-appo-vtrace.yaml	[RLlib] Upgrade gym version to 0.21 and de	nths ago
halfcheetah-appo.yaml	[RLlib] Refactor: All tf static graph code sho	nths ago
halfcheetah-ppo.yaml	[RLlib] Auto-framework, retire `use_pytorch`	ears ago
hopper-ppo.yaml	[RLlib] Auto-framework, retire `use_pytorch`	ears ago
humanoid-ppo-gae.yaml	[RLlib] Auto-framework, retire `use_pytorch` in favor of `framework=....	2 years ago

20 lines (20 sloc) | 506 Bytes

```yaml
cartpole-appo:
    env: CartPole-v0
    run: APPO
    stop:
        episode_reward_mean: 150
        timesteps_total: 200000
    config:
        # Works for both torch and tf.
        framework: tf
        num_envs_per_worker: 5
        num_workers: 1
        num_gpus: 0
        observation_filter: MeanStdFilter
        num_sgd_iter: 6
        vf_loss_coeff: 0.01
        vtrace: false
        model:
            fcnet_hiddens: [32]
            fcnet_activation: linear
            vf_share_layers: true
```

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

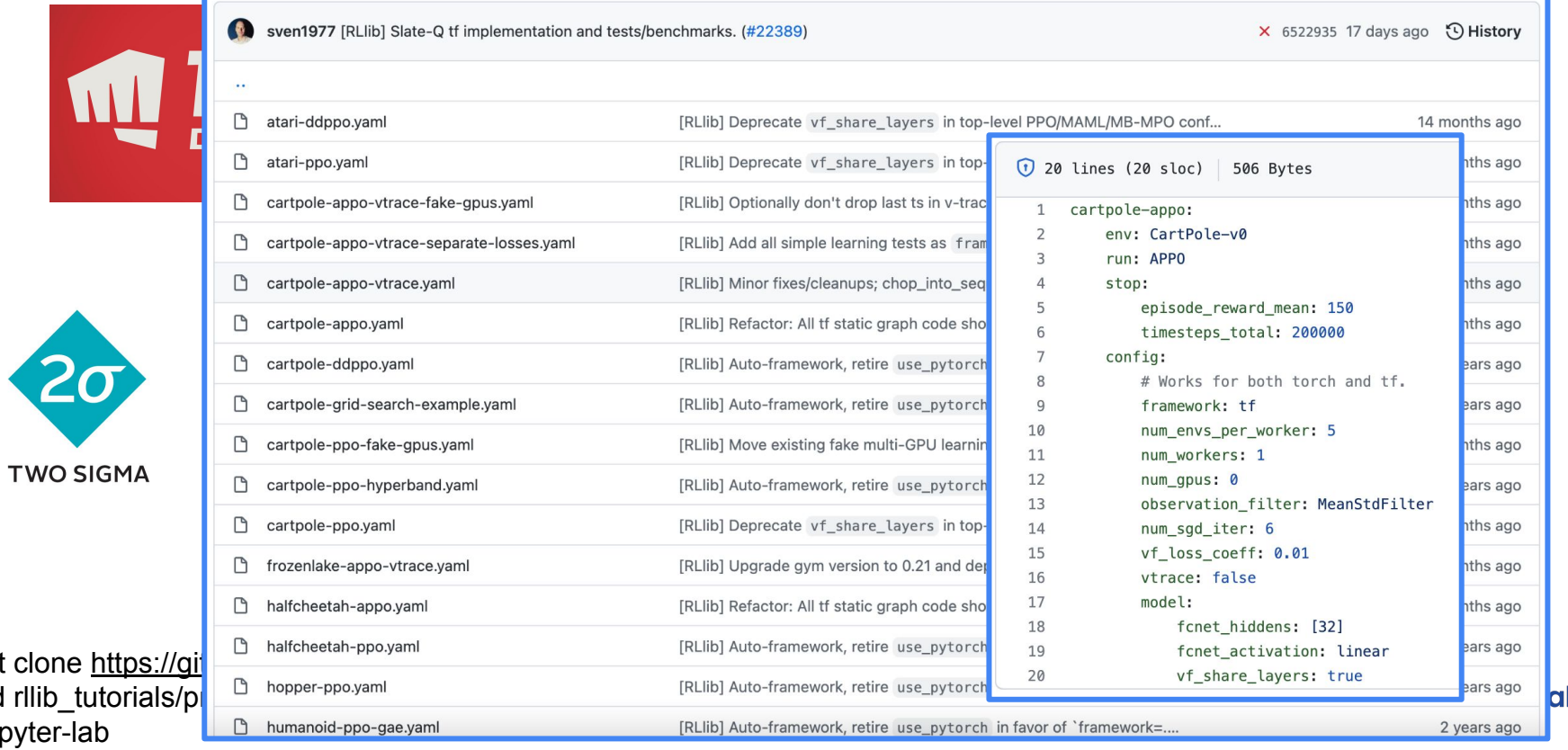
ale

And why should I use RLlib?

Because these companies here do!

Thx for presen… Summits!

Tuned examples for most of our algorithms on popular environments.

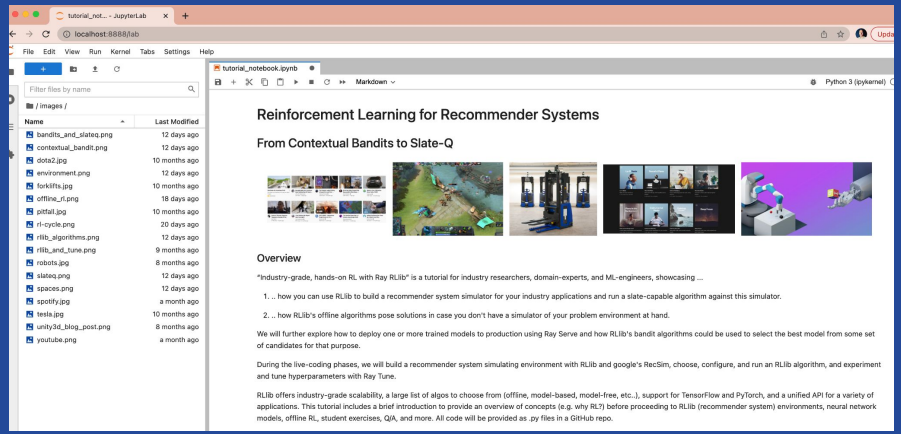


TWO SIGMA

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

…ale

15 min Break :)
Then … moving to our Jupyter Notebook



anyscale

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**

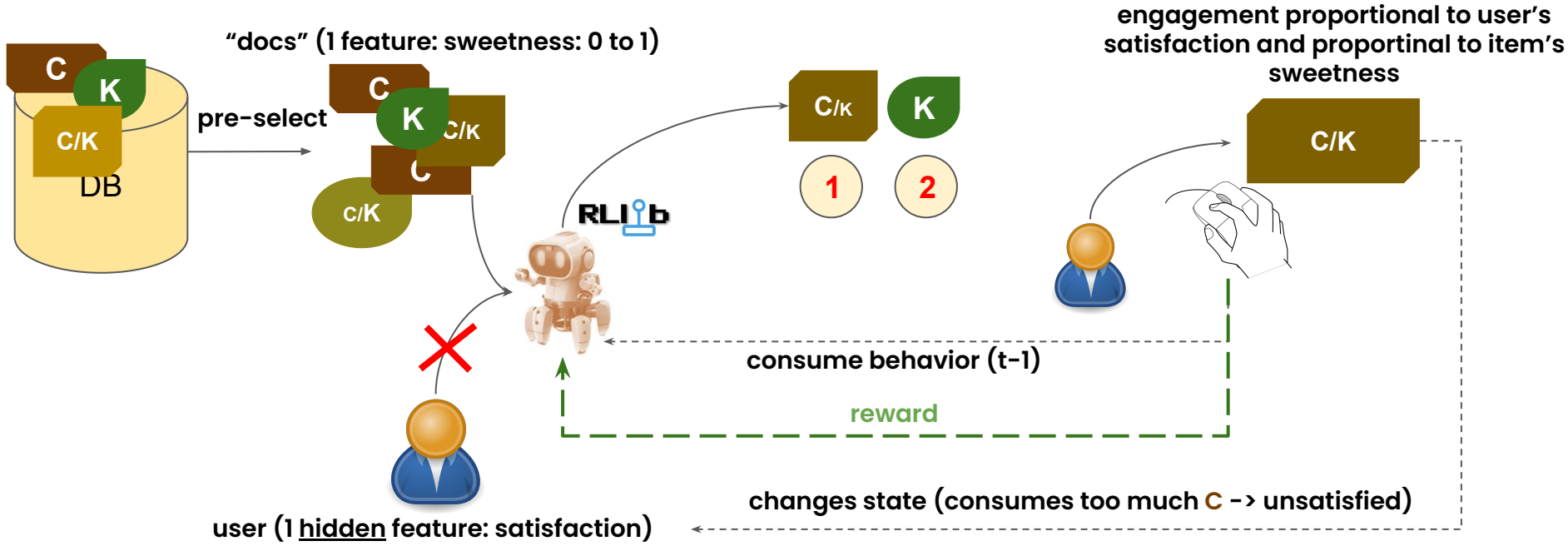
anyscale

Google RecSim

The "Long Term Satisfaction" Problem

Kale: Sweetness=0.0
Low engagement; high satisfaction

Chocolate: Sweetness=1.0
High engagement; low satisfaction



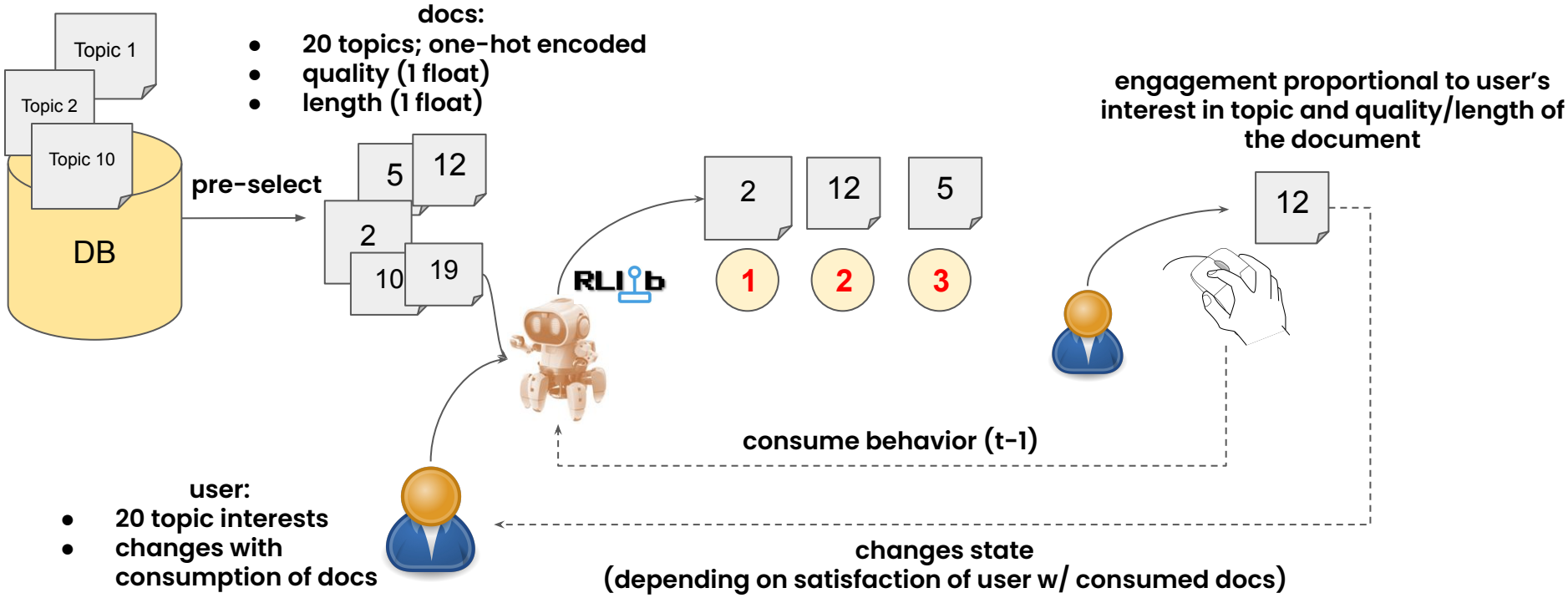
https://github.com/google-research/recsim

```
$ pip install recsim
```

anyscale

Google RecSim

The "Interest Evolution" Problem



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

```
$ pip install recsim
```

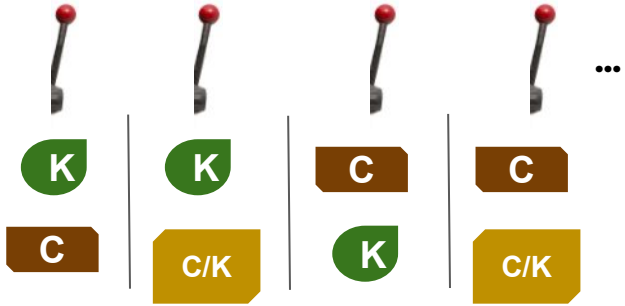
anyscale

Contextual n-armed Bandit

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

exploration/exploitation

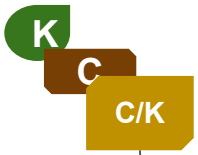
arms (flattened action space)



pick 1 arm (action)

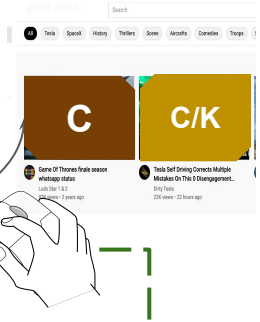
n candidate docs (vectors)

"context" (observation)



RLlib

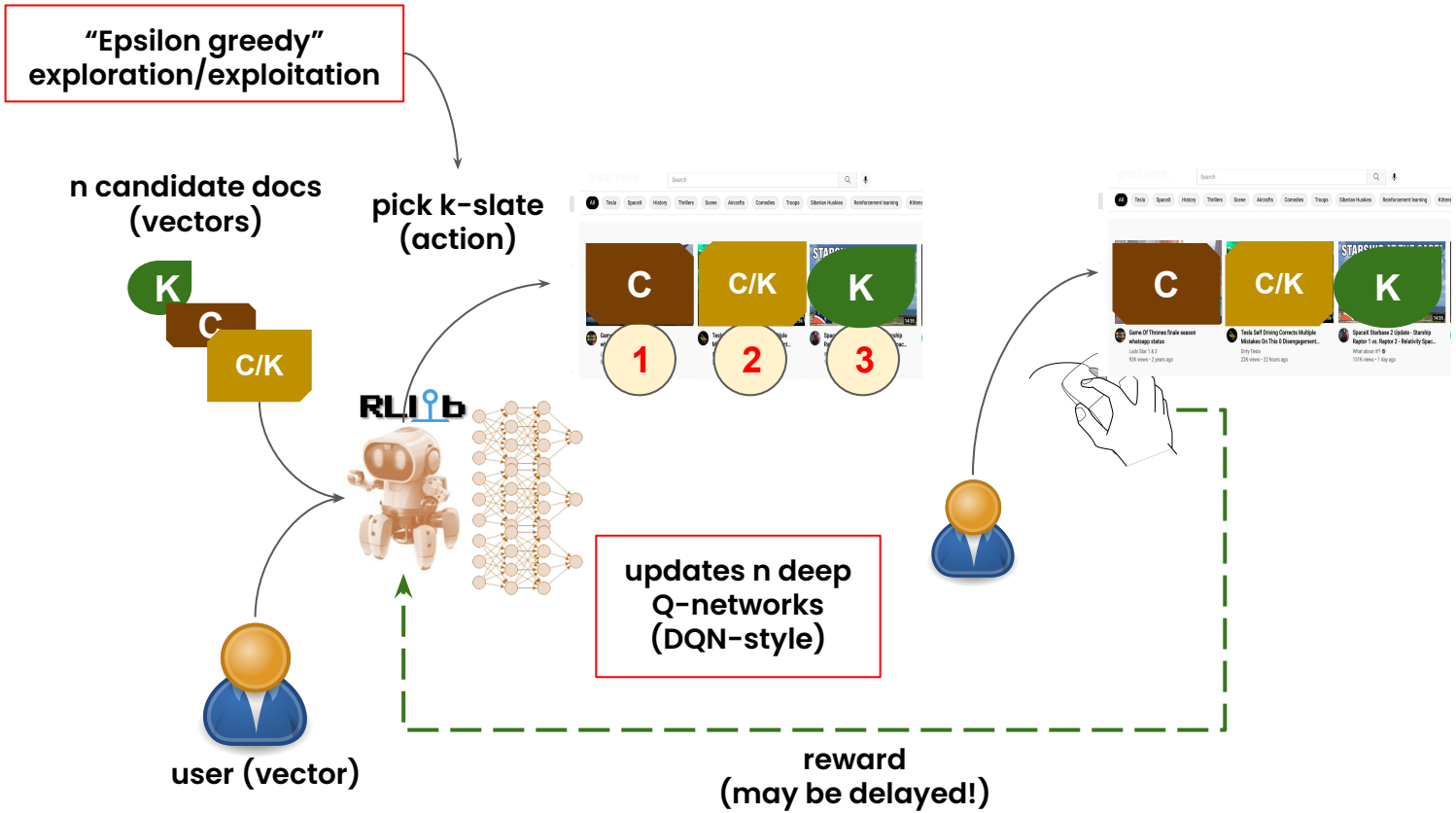
updates linear reward model (after each timestep; "online")



reward (must be immediate)

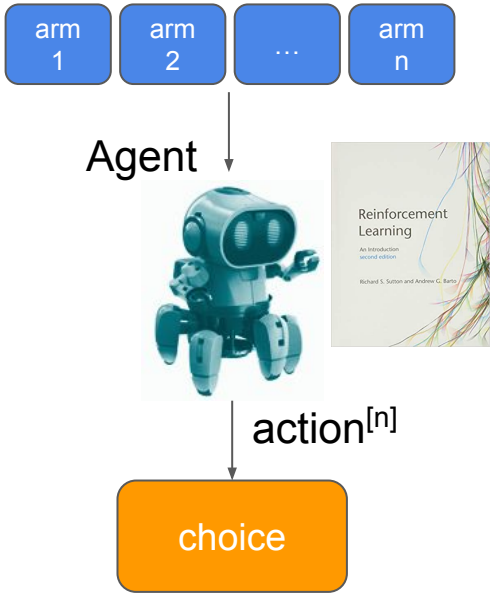
anyscale

Slate-Q Algorithm



anyscale

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



Bandit

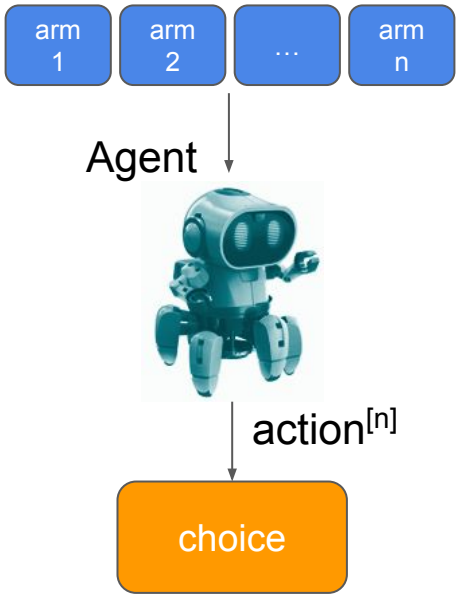
One-armed Bandit

n-armed Bandit

anyscale

RLlib

Simple, n-Armed Bandit



n-armed Bandit

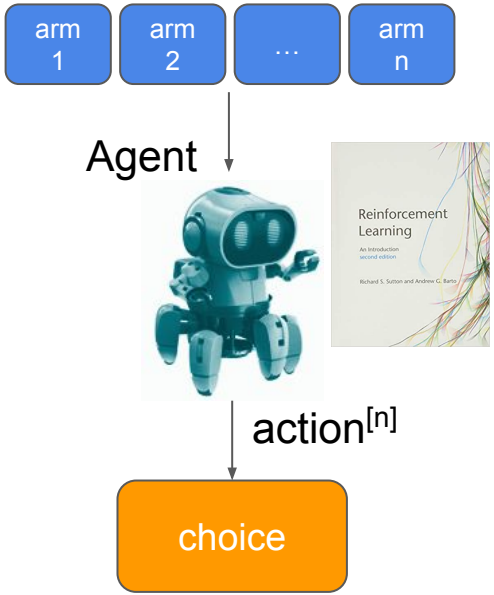
- Models each arm's reward signal ("Q-function").
- Needs to carefully balance exploration (i.e. pick an arm that currently has a high Q-estimate, but also still lots of uncertainty (optimism in the face of uncertainty), e.g. UCB sampling:

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

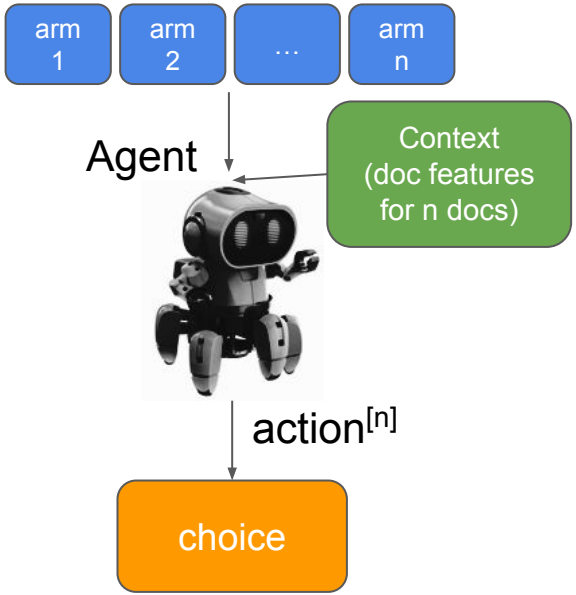
- Received rewards are used to update Q-function estimates for the picked arm.
- No "context"; Each arm always uses same underlying (fixed) model.

anyscale

Contextual Bandit



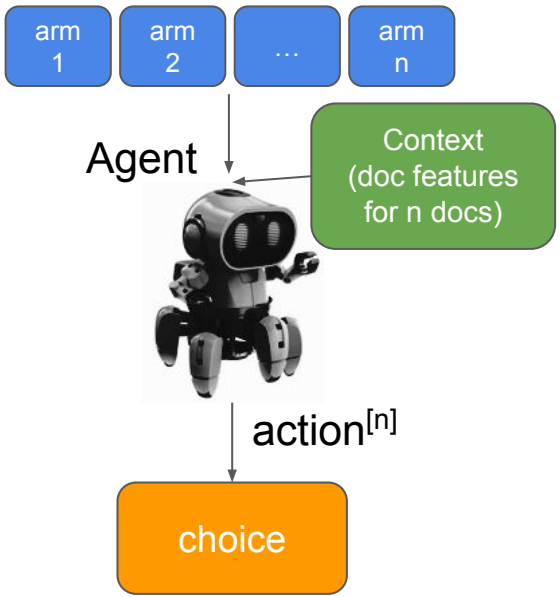
n-armed Bandit



contextual n-armed Bandit

anyscale

Contextual Bandit



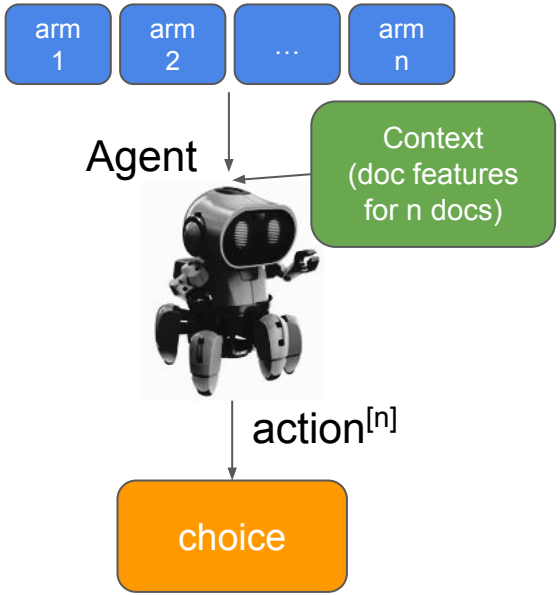
contextual n-armed Bandit

- 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.

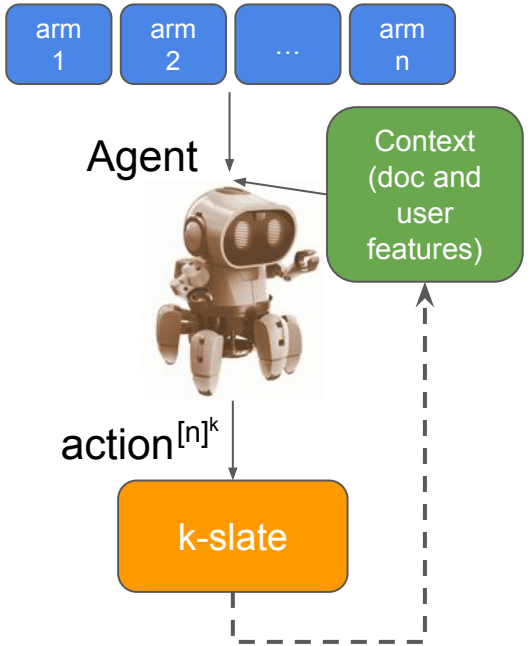
anyscale

RLlib

Slate-Q



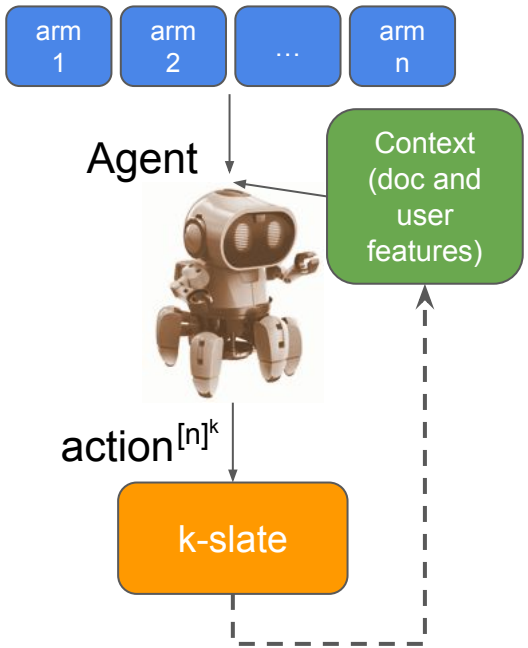
contextual n-armed Bandit



Slate-Q

anyscale

Slate-Q



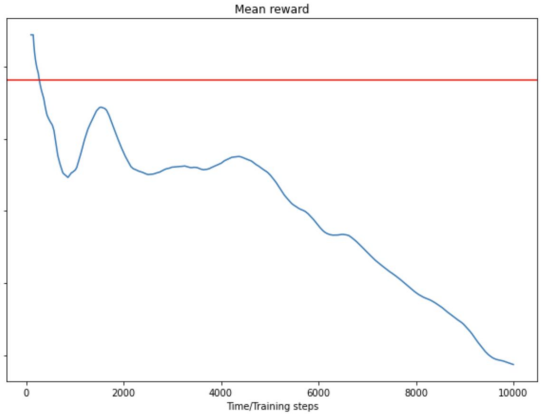
Slate-Q

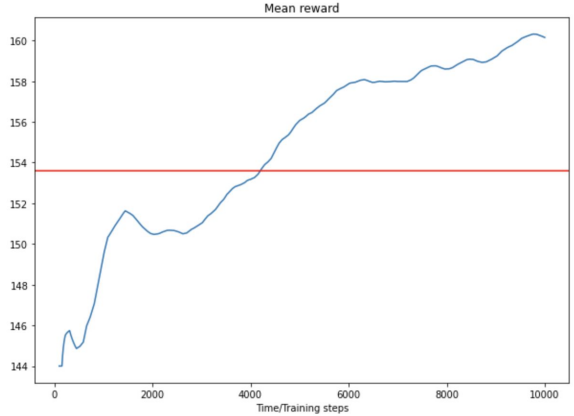
- Sees both “doc” and “user” information and can thus learn to model different users.
- Policy is a set of n Q-heads, where n is the number of document candidates to pick from. Input to each of these n heads is the user vector + the respective doc vector.
- Exploration: Epsilon-greedy.
- Updates: Analogous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

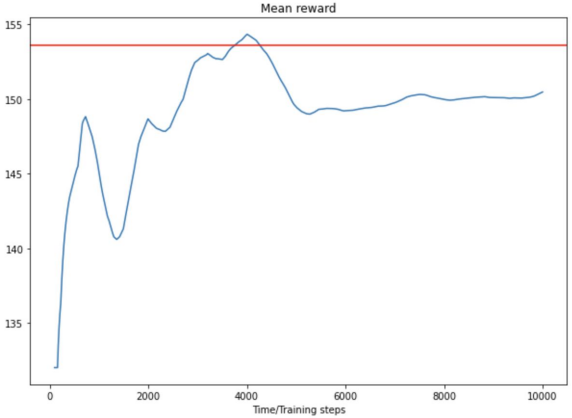
anyscale

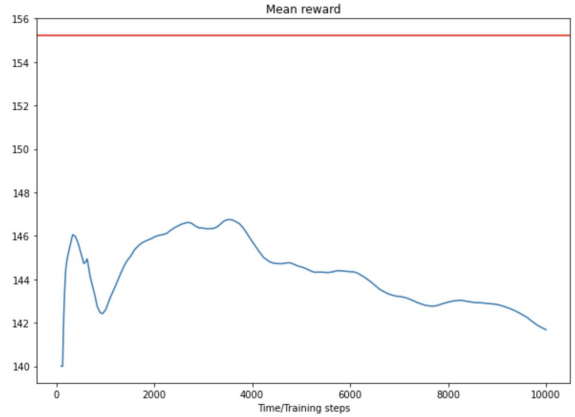
Contextual UCB Bandit Results

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

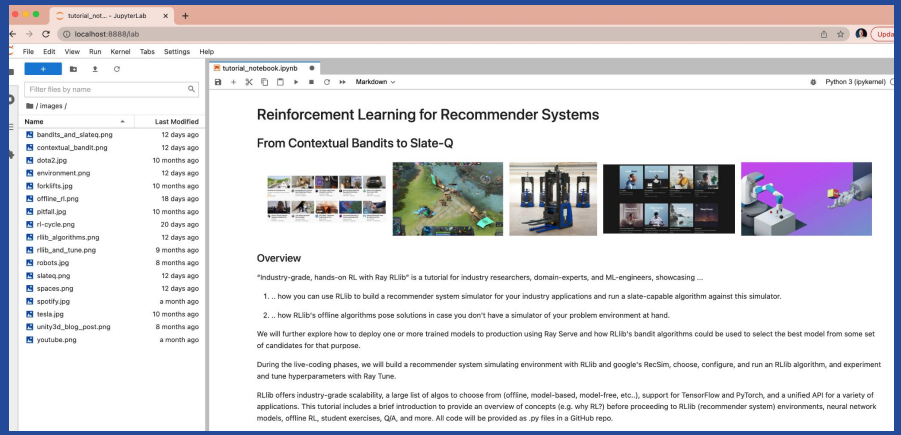








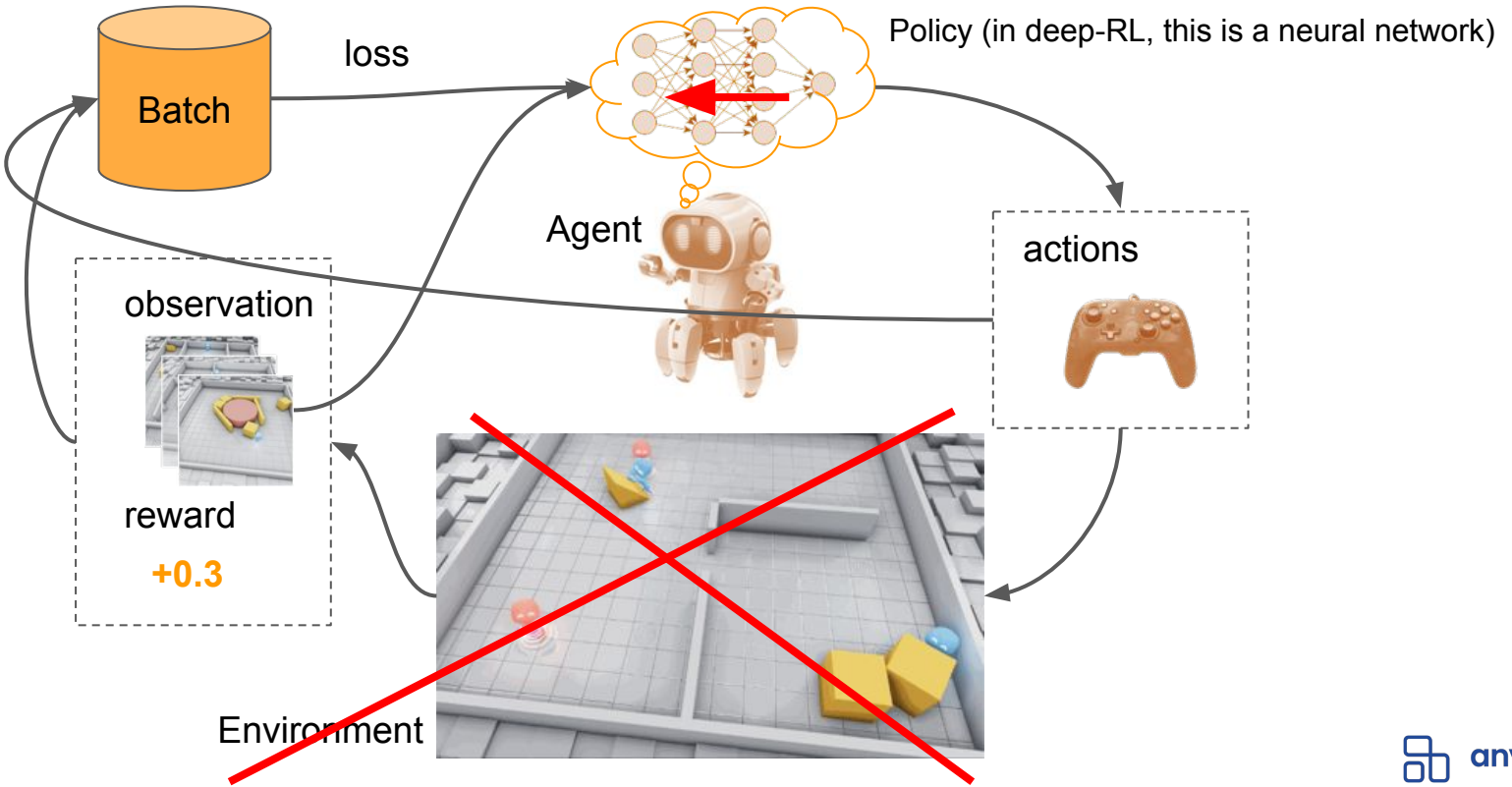
And now …
Moving to our Jupyter Notebook



anyscale

What's Offline Reinforcement Learning?

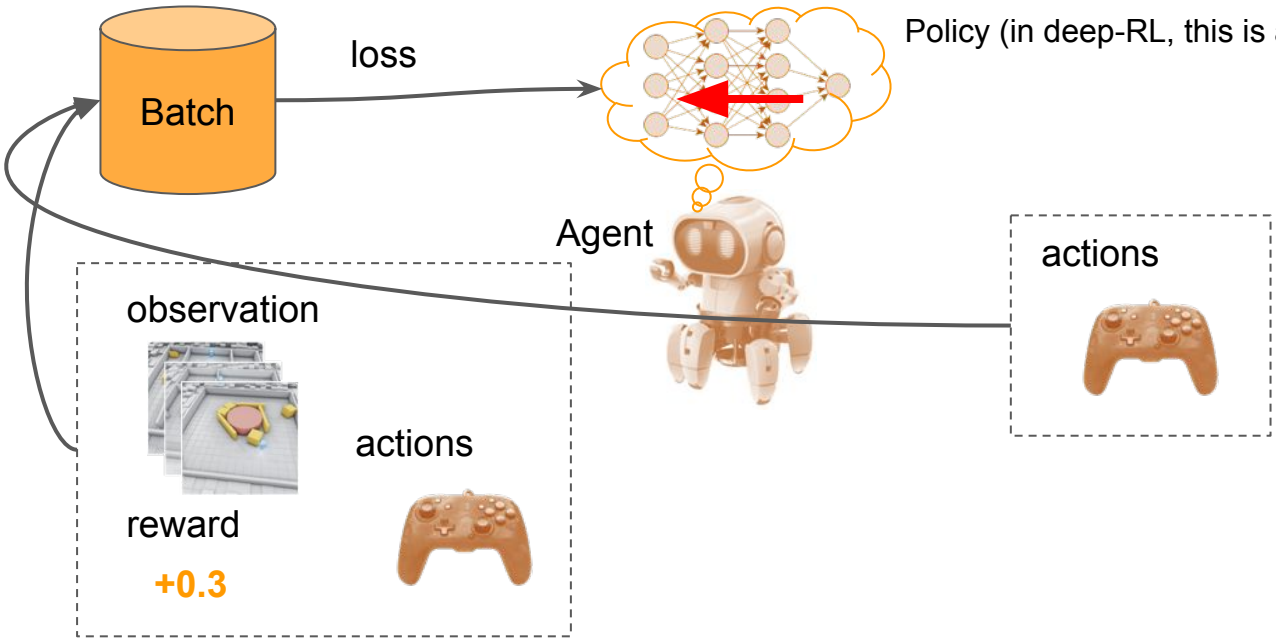
Aka: "Batch RL"



anyscale

What's Offline Reinforcement Learning?

Aka: Batch RL

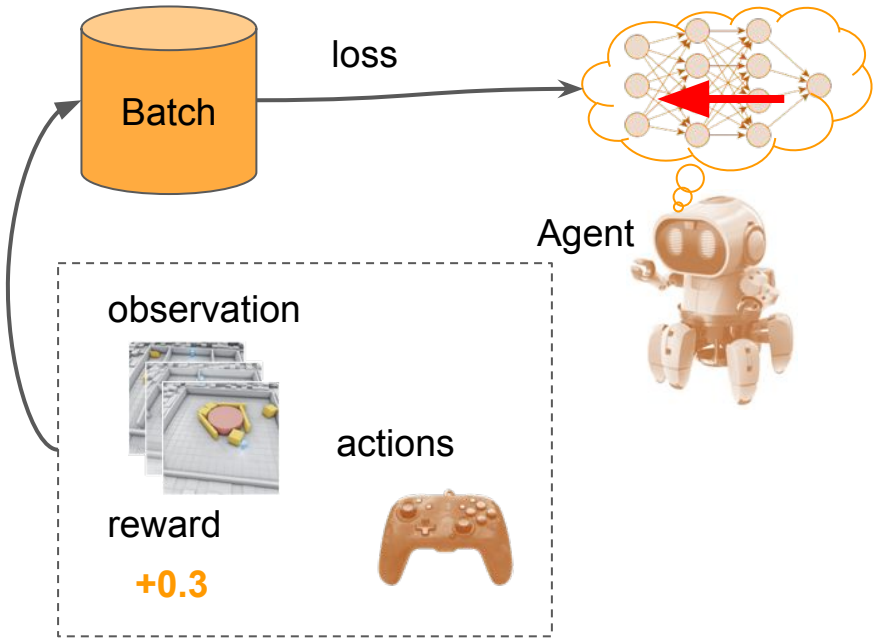


Historic data recorded in the past (e.g. a JSON file!).

anyscale

What's Offline Reinforcement Learning?

Aka: Batch RL



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

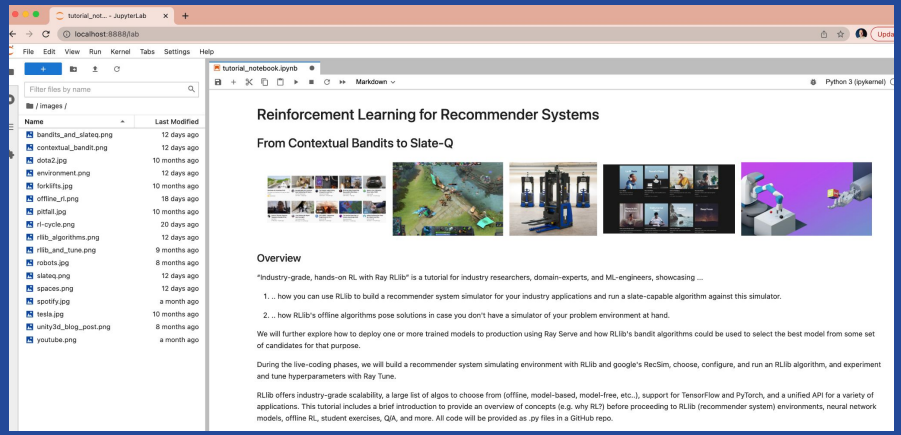
Policy (in deep-RL, this is a neural network)

<u>2 ways of doing this:</u>

- Behavioral Cloning (BC), aka: "Imitation learning": 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.

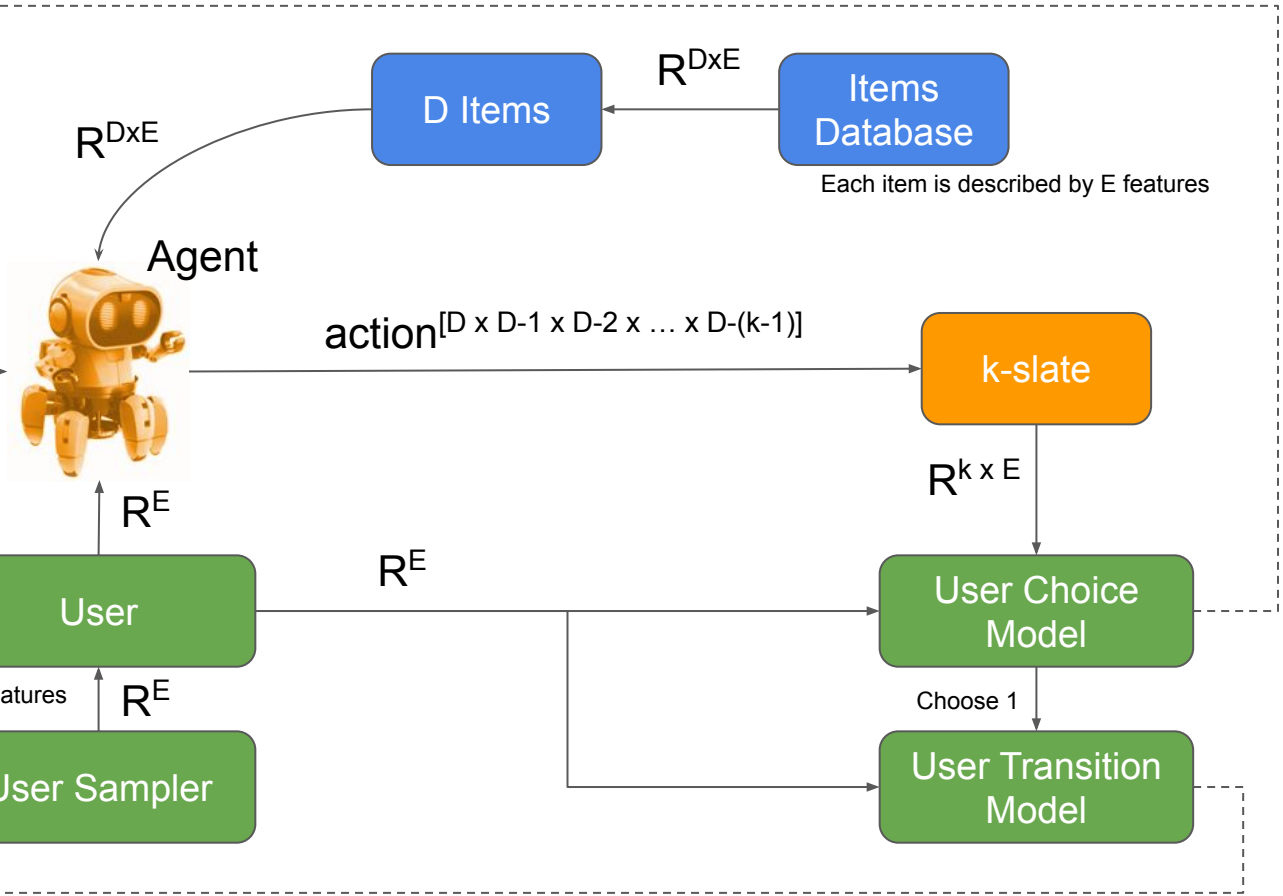
anyscale

And now …
Moving to our Jupyter Notebook



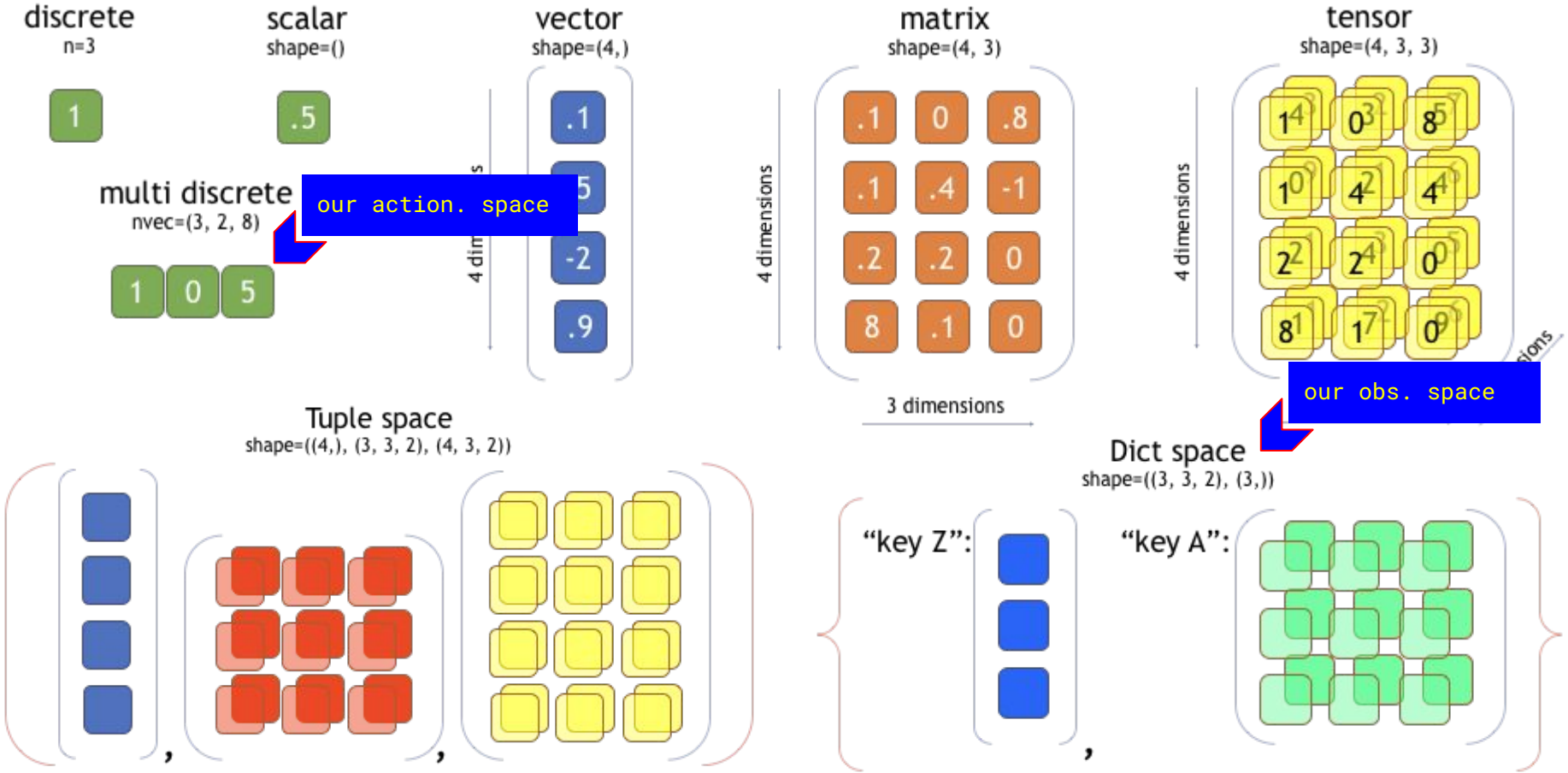
anyscale

Our Environment - “RecommSys-001” 😶



anyscale

What's a Space?



Quick demo of notebook