Deep Learning

CSCI 1470

Eric Ewing

Day 33: PPO, RLHF, and AGI

Friday, 4/18/25

On-Policy and Off-Policy Learning

RL algorithms collect experiences and learn from these experiences

On-Policy Algorithms have to collect experiences with the policy they are learning

Off-Policy Algorithms can use **any** policy to collect experiences

Review: On + Off-Policy Learning

	On-Policy	**Off Policy**
Summary	Learns policy/value function based on policy used during training	Learns policy independent of policy used to collect experiences during training
Algorithms	SARSA, Policy Gradient, Actor Critic, PPO	Q-Learning, Off-policy Actor-Critic, Deep Deterministic Policy Gradient (DDPG)

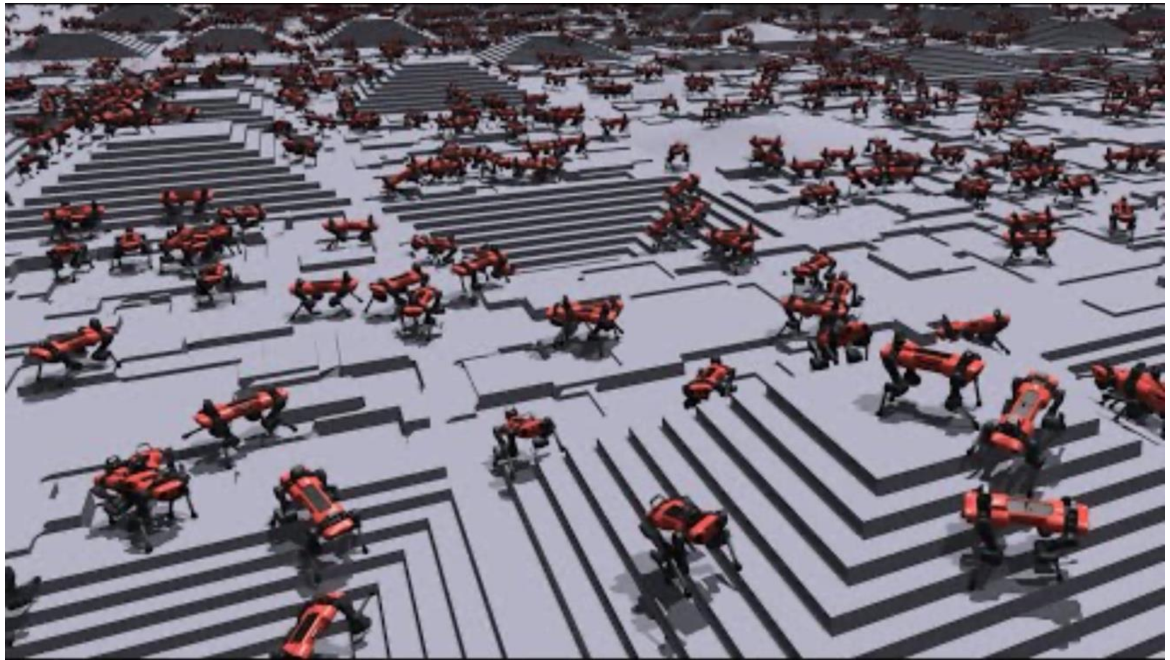
Off-Policy Learning

Most of the time in RL, collecting the data is computationally expensive.

Maybe we shouldn't throw away useful data immediately…

So far, we've looked at an example, learned from it, and discarded it.

In all our other problems, we always learned from data multiple times (i.e., epochs)

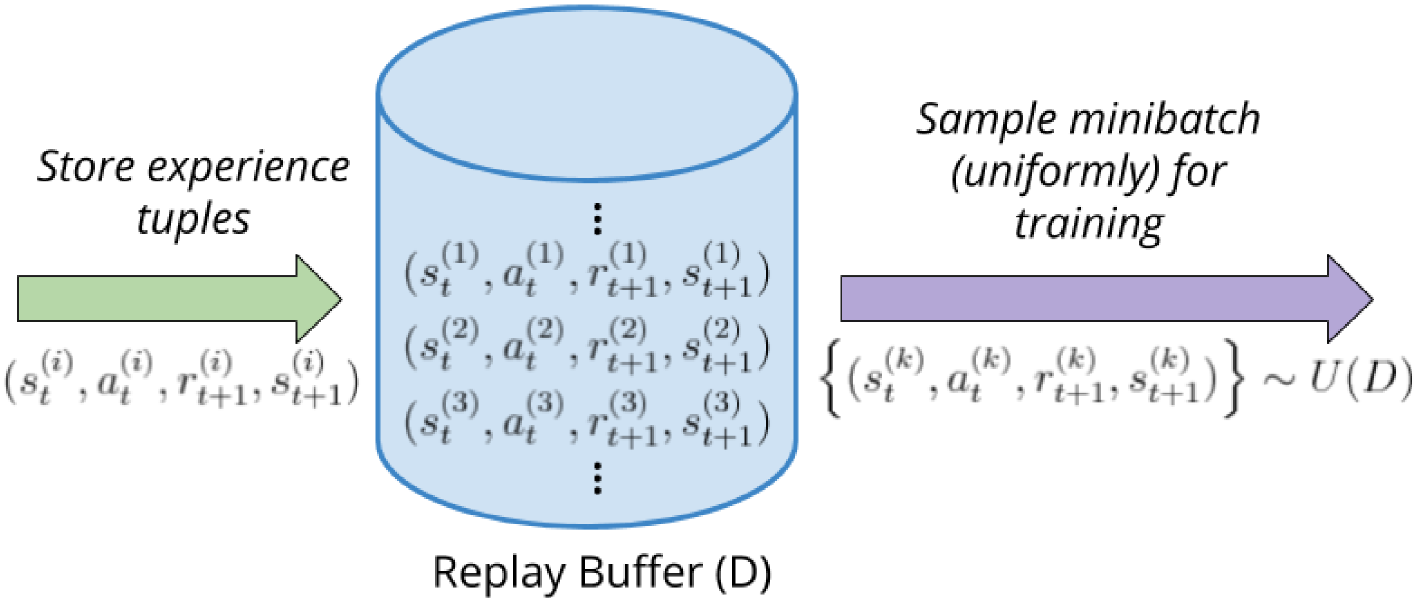


Experience Replay and Replay Buffers

Keep a memory of experiences (state, action, reward, next_state)

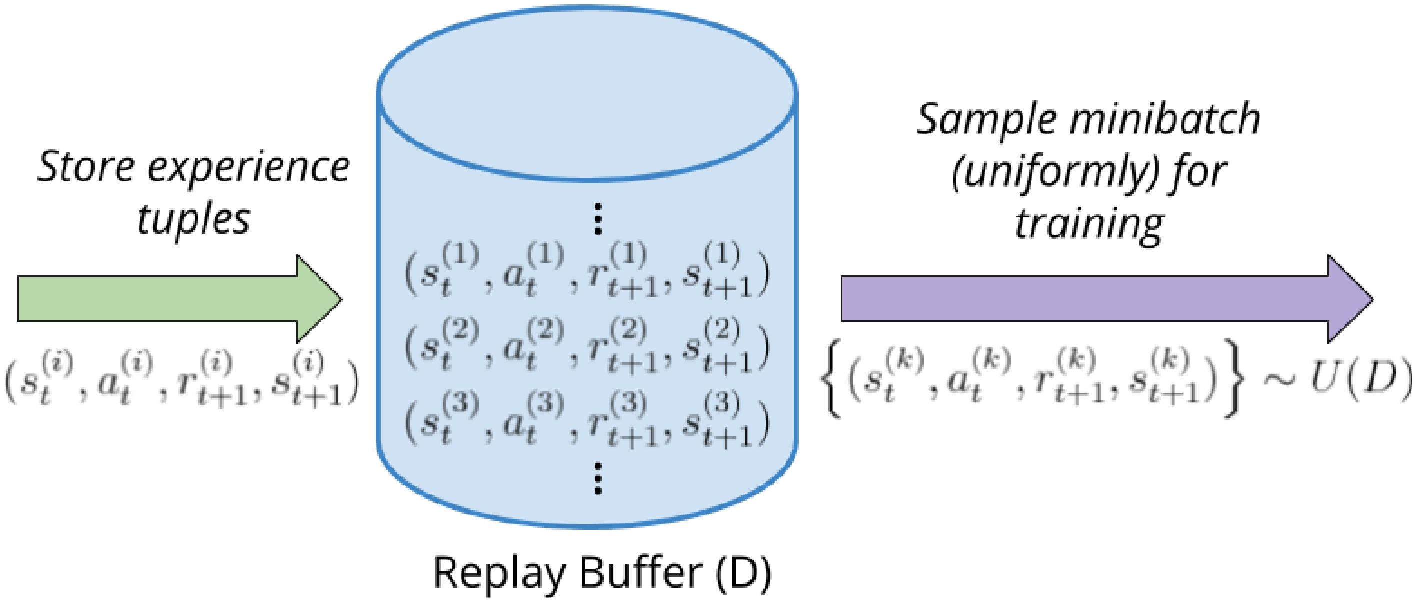
As you collect new experiences, remove oldest experiences from buffer

To train model, sample batch of data from buffer



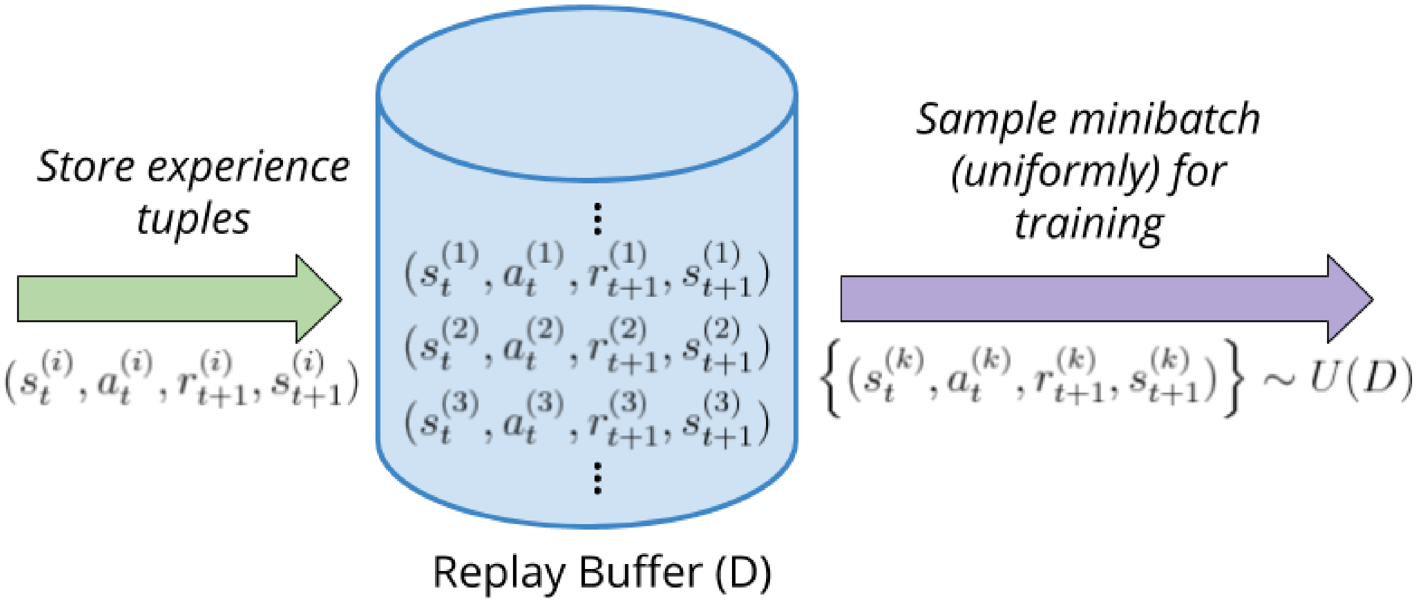
On-Policy Learning

Can we use Replay Buffers with On-Policy learning algorithms (e.g., REINFORCE, Actor-Critic, etc.)?



On-Policy Learning

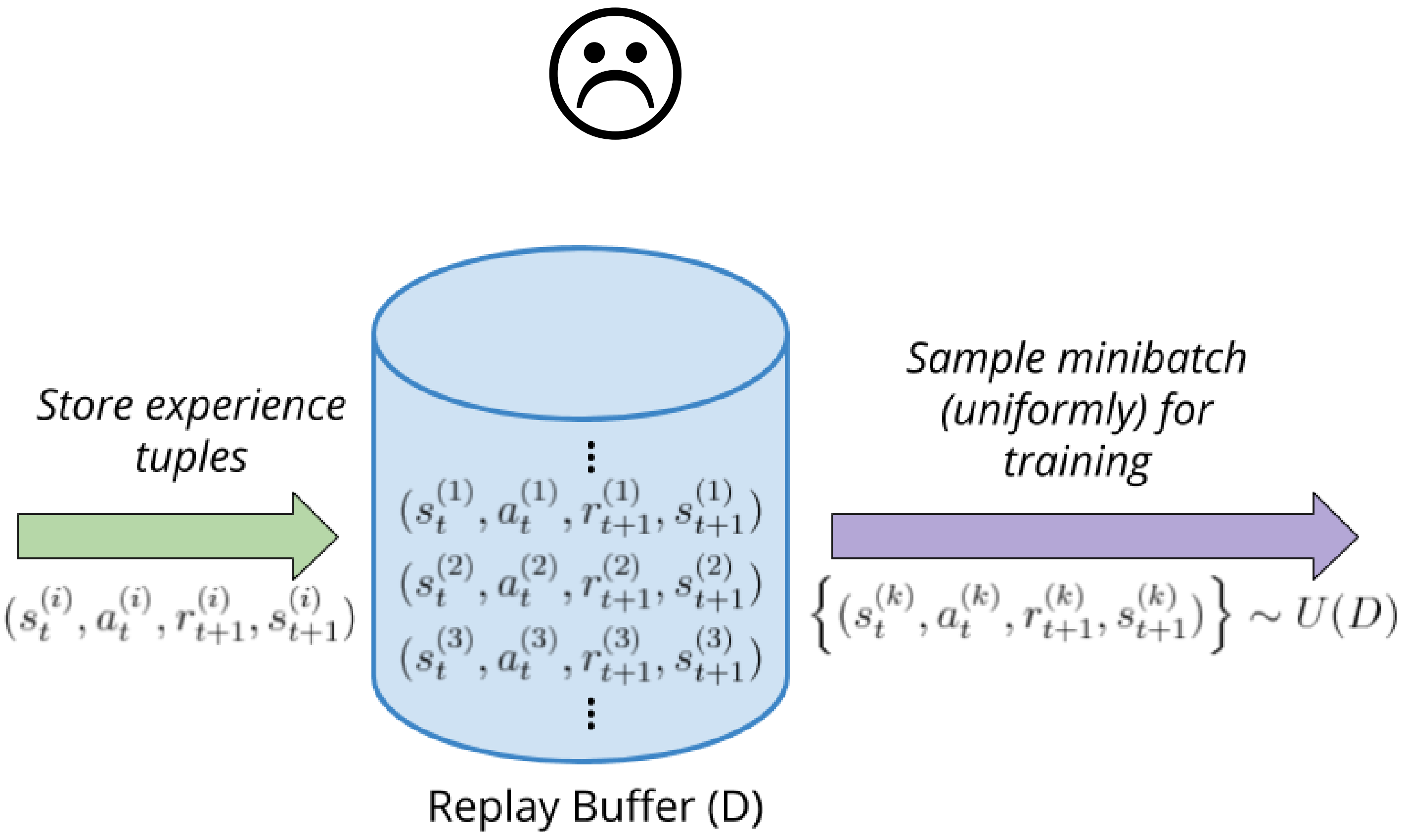
Can we use Replay Buffers with On-Policy learning algorithms (e.g., REINFORCE, Actor-Critic, etc.)?



No! Data in the buffer was collected with an older policy and we can only learn on experiences collected using the current policy...

On-Policy Learning

Can we use Replay Buffers with On-Policy learning algorithms (e.g., REINFORCE, Actor-Critic, etc.)?



No! Data in the buffer was collected with an older policy and we can only learn on experiences collected using the current policy…

But what if we actually could…

Off-Policy Policy Gradient:

Data collected under policy $\beta(a|s)$ (i.e., older version of policy)

We can re-weight our gradient according to the old policy:

$$\rho = \frac{\pi(a|s)}{\beta(a|s)}$$

$$\nabla_\theta J(\theta) = \sum_{(s,a)\in batch} \rho \cdot Q^\pi(s,a) \nabla_\theta \ln \pi(s,a)$$

How much should we weigh each experience?

Actor-Critic with Importance Sampling

Off-Policy Actor Critic: https://arxiv.org/pdf/1205.4839

But what if we actually could...

Off-Policy Policy Gradient:

Data collected under policy $\beta(a|s)$ (i.e., older version of policy)

We can re-weight our gradient according to the old policy:

$$\rho = \frac{\pi(a|s)}{\beta(a|s)}$$

$$\nabla_\theta J(\theta) = \sum_{(s,a) \in batch} \rho \cdot Q^\pi(s,a) \nabla_\theta \ln \pi(s,a)$$

Actor-Critic with Importance Sampling

Store action probabilities $\beta(a|s)$ in replay buffer

Off-Policy Actor Critic: https://arxiv.org/pdf/1205.4839

Trust Region Policy Optimization

Insight: the reason that variance is bad is that it can cause large updates to π_θ

Add a constraint to how large of an update can be applied:

KL-Divergence between old and new policy must be below some hyperparameter Δ

TRPO works well and has lower variance during training, but it's painfully complicated.
Inverting a Hessian introduces numerical precision errors that need to be avoided.
Can we come up with something simpler?

$$\mathcal{L}(\theta_k,\theta) = \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}}\left[\frac{\overset{\textit{new}}{\pi_\theta(a|s)}}{\overset{\textit{old}}{\pi_{\theta_k}(a|s)}} A^{\pi_{\theta_k}}(s,a)\right]$$

Advantage function (or TD-Error)

Gradient incorporating constraint:

$$\sqrt{\frac{2\delta}{g^T H^{-1} g}}\, H^{-1} g$$

H is the *Hessian*, i.e., 2nd order partial derivatives, g is the gradient

Paper: https://arxiv.org/pdf/1502.05477

Proximal Policy Optimization

TRPO is complicated…

What if instead of constraining the update with KL-Divergence, we clipped the update if it's too big…

$$\rho_{clipped} = clip[\frac{\pi^{new}(a|s)}{\pi^{old}(a|s)}, 1-\epsilon, 1+\epsilon]$$

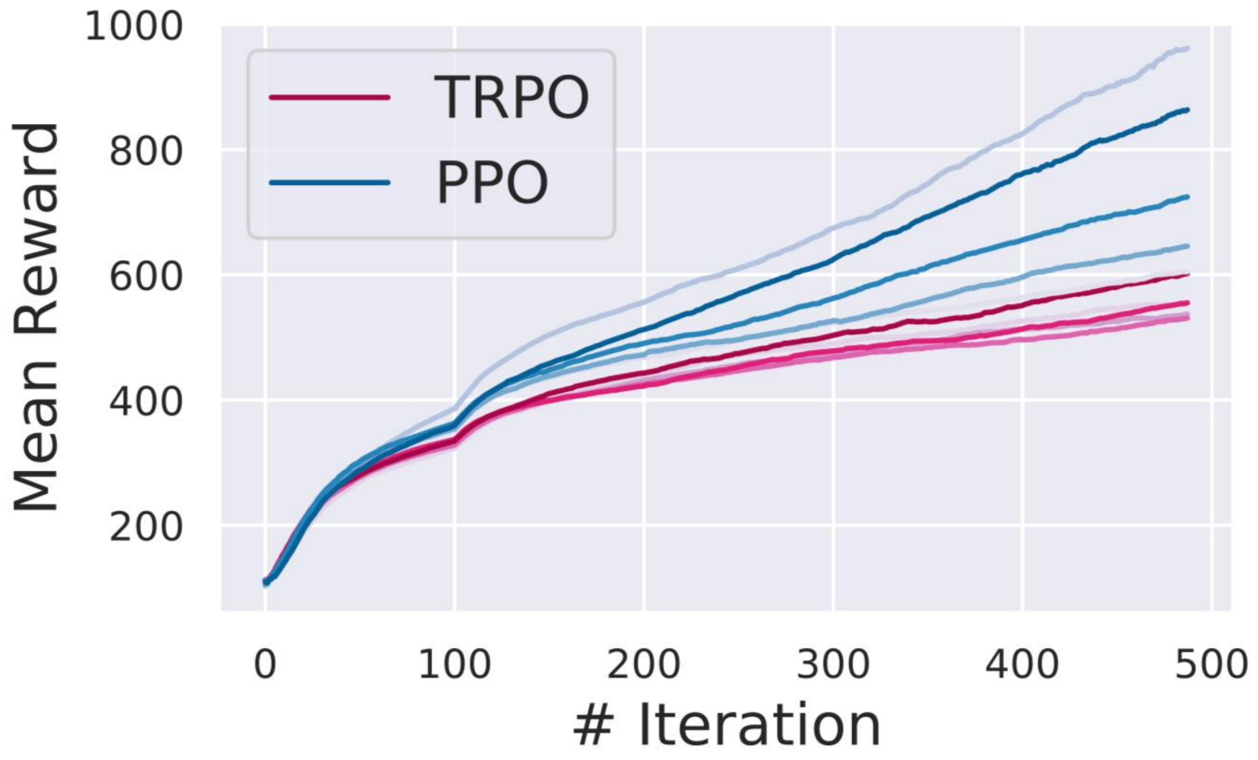
$$J^{PPO}(\theta) = \mathbb{E}[\min(\rho_{clipped} \cdot \left(r + \gamma V^{\pi^{old}}(s') - V^{\pi^{old}}(s)\right), \rho\left(r + \gamma V^{\pi^{old}}(s') - V^{\pi^{old}}(s)\right)]$$

Spinning Up PPO: https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

PPO is (basically) State-Of-The-Art (SOTA)

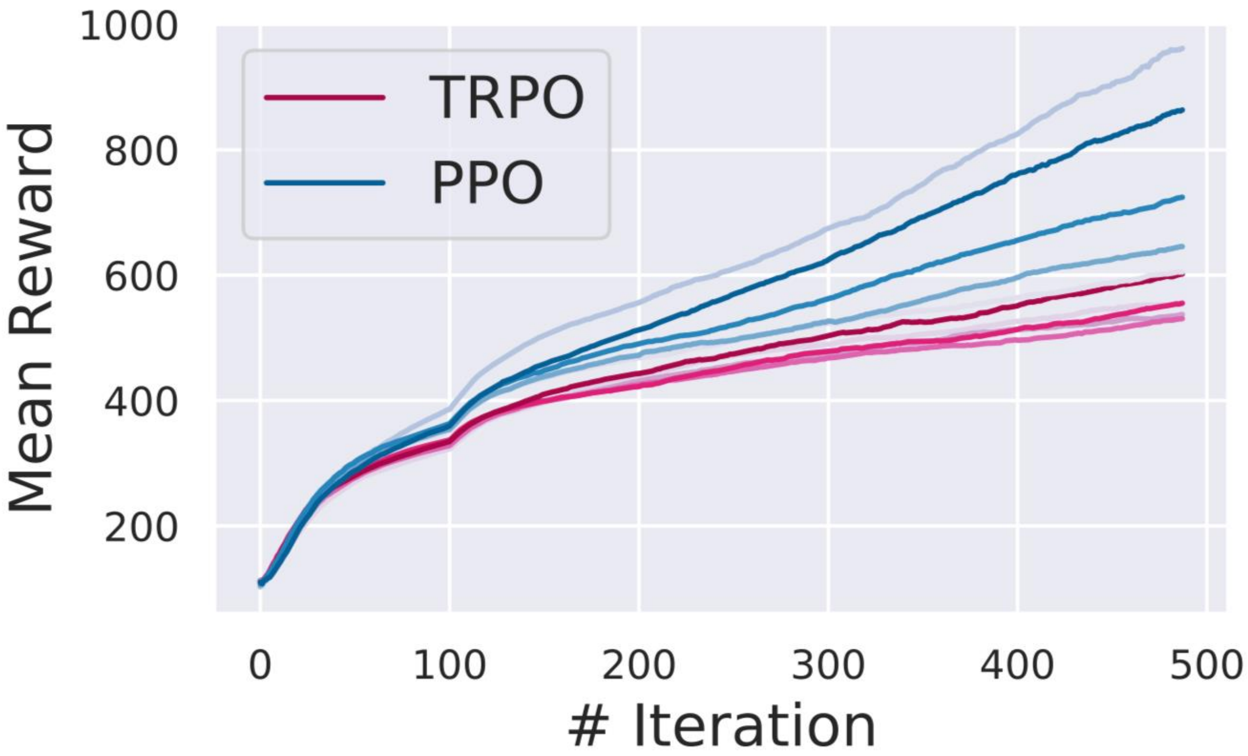
Provides **fast**, **sample-efficient**, and **stable** training



PPO

PPO is (basically) State-Of-The-Art (SOTA)

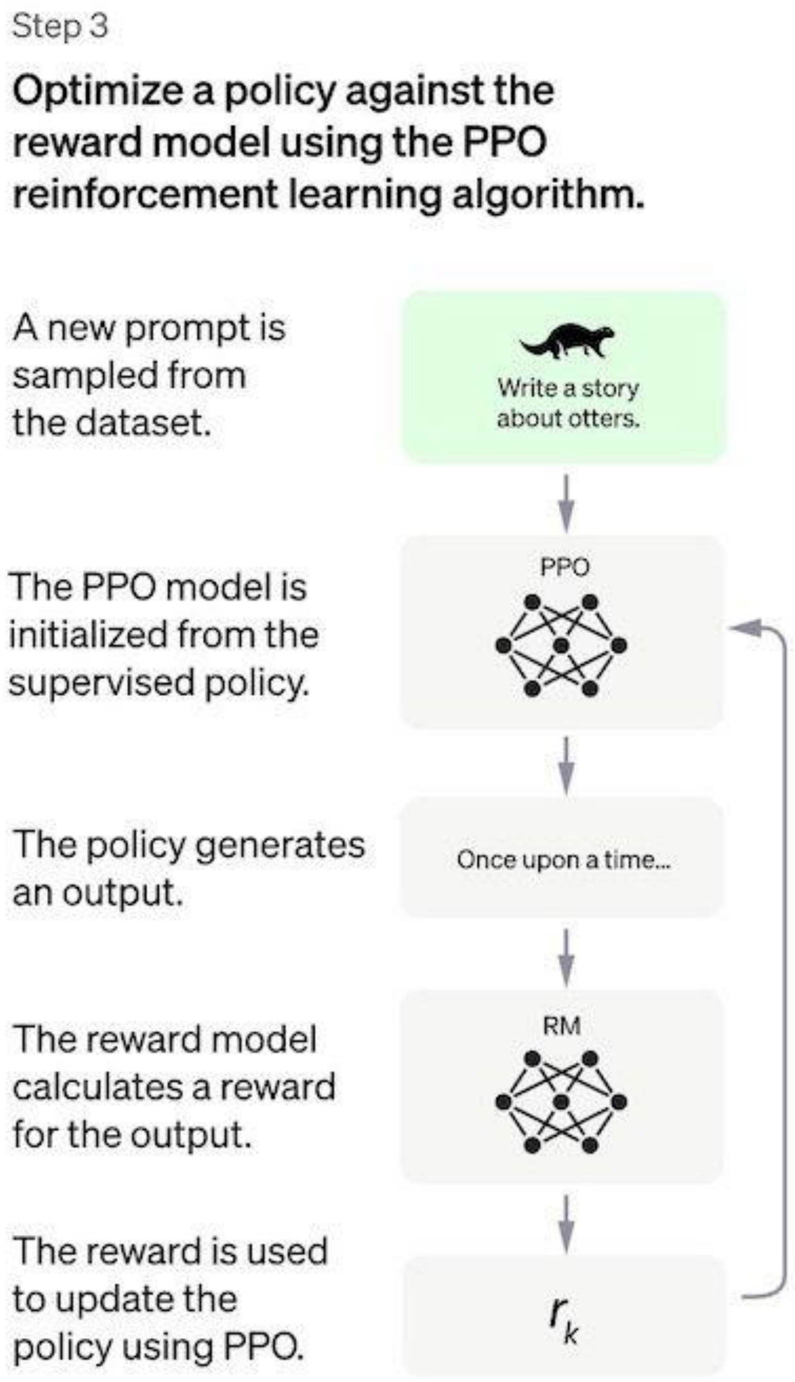
Provides **fast**, **sample-efficient**, and **stable** training



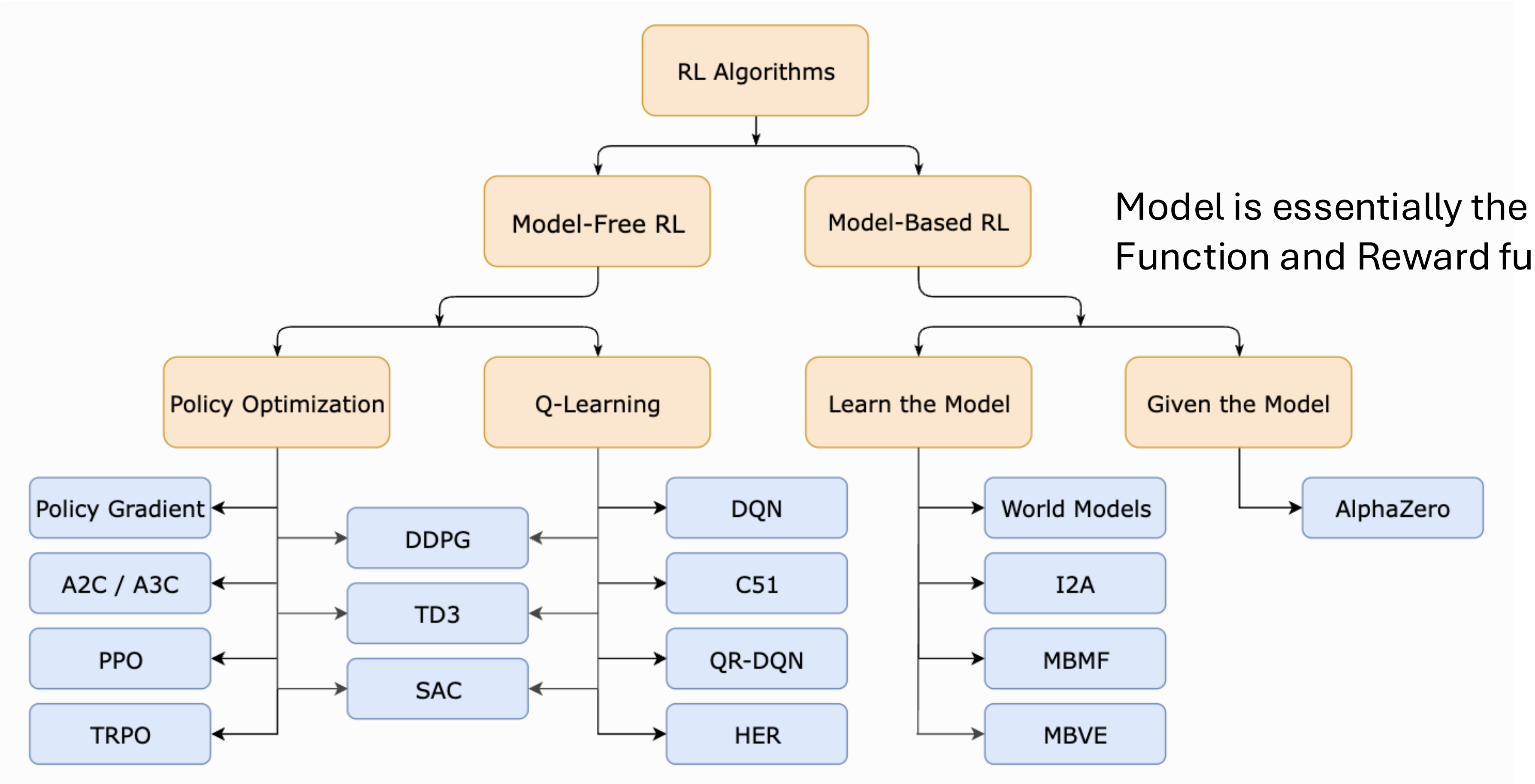
PPO: OpenAI5

PPO

Final phase of training ChatGPT



RL Hierarchy



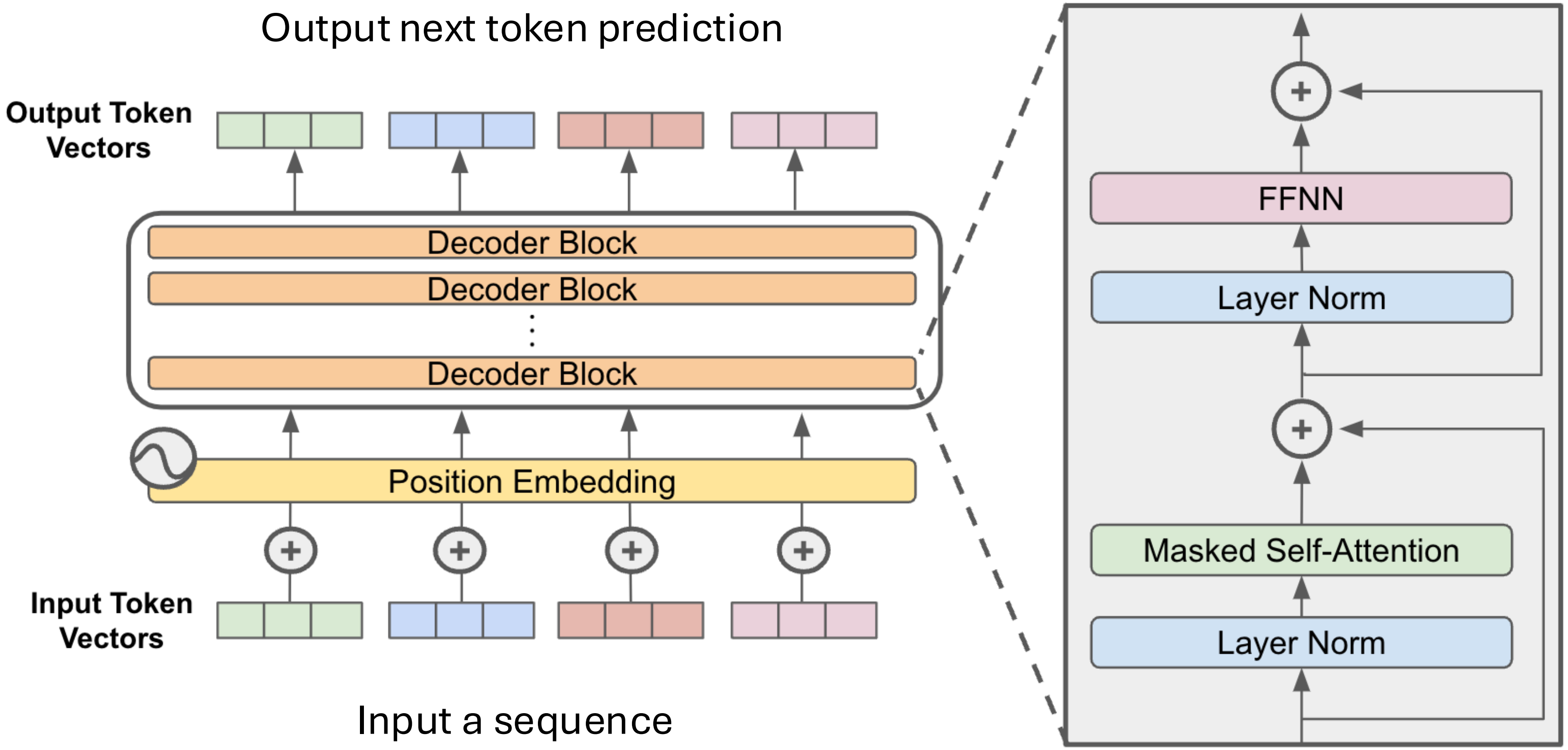
Model is essentially the Transition Function and Reward function?

Source and helpful explanations: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Language Modelling Revisited

Typically framed as supervised learning-style problem:

1. Given some context (e.g., a question)
2. Predict the next token.



Turning Language modelling into an MDP

MDP: <S, A, P, R, γ>

States: Each state is a sequence of tokens

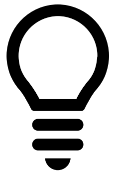
Actions: LLM adds the next token

Transition Function: Transitions are deterministic, given a state and next token, the next state is just the token appended to the previous state

Reward Function: The LLM should be rewarded for good responses, but how do we know what the quality of response is?

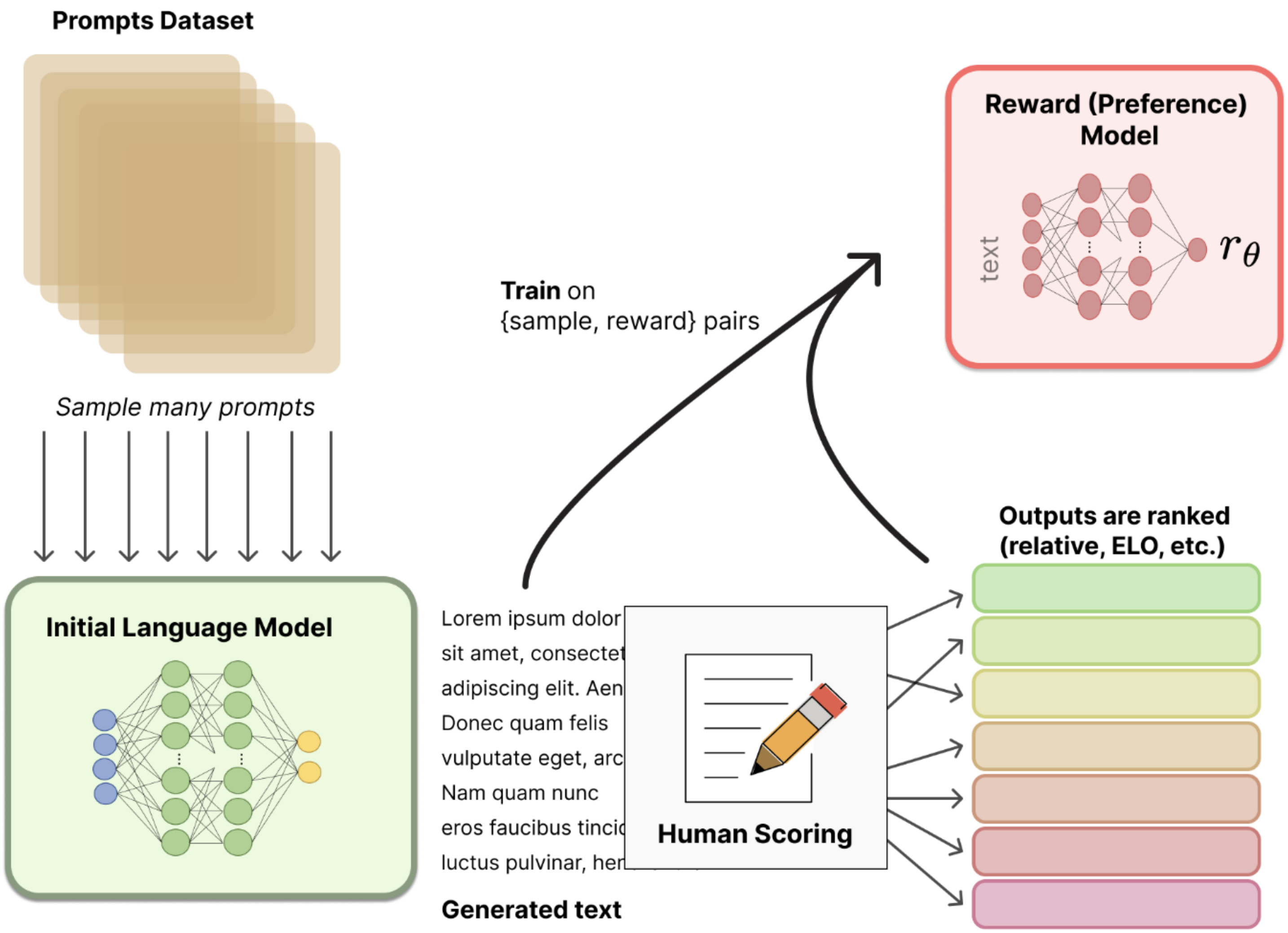
Reward Modeling

In MDPs, the reward function is a mapping from states to rewards



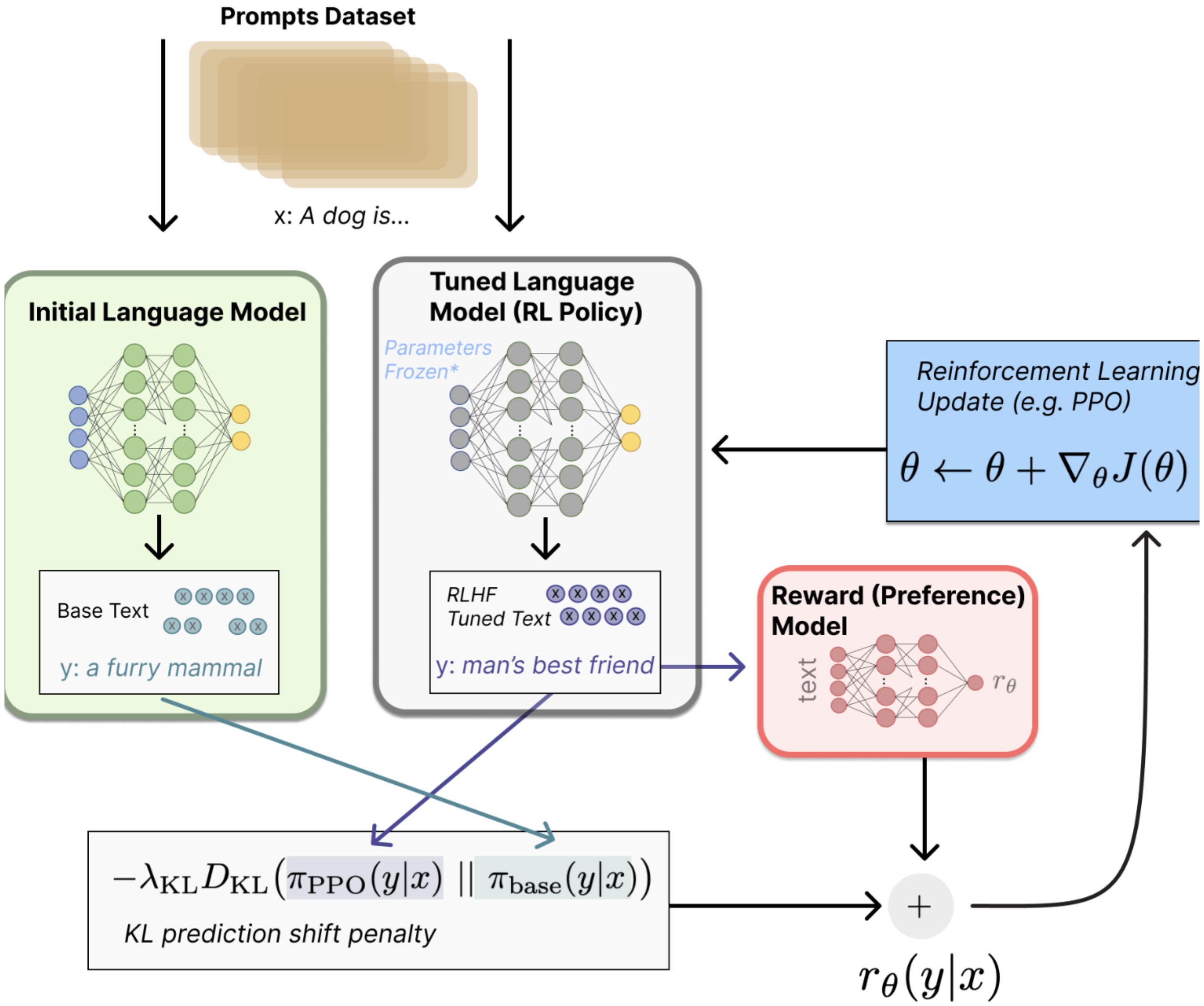
Reward Modeling: Learn a reward function

Reward Modeling



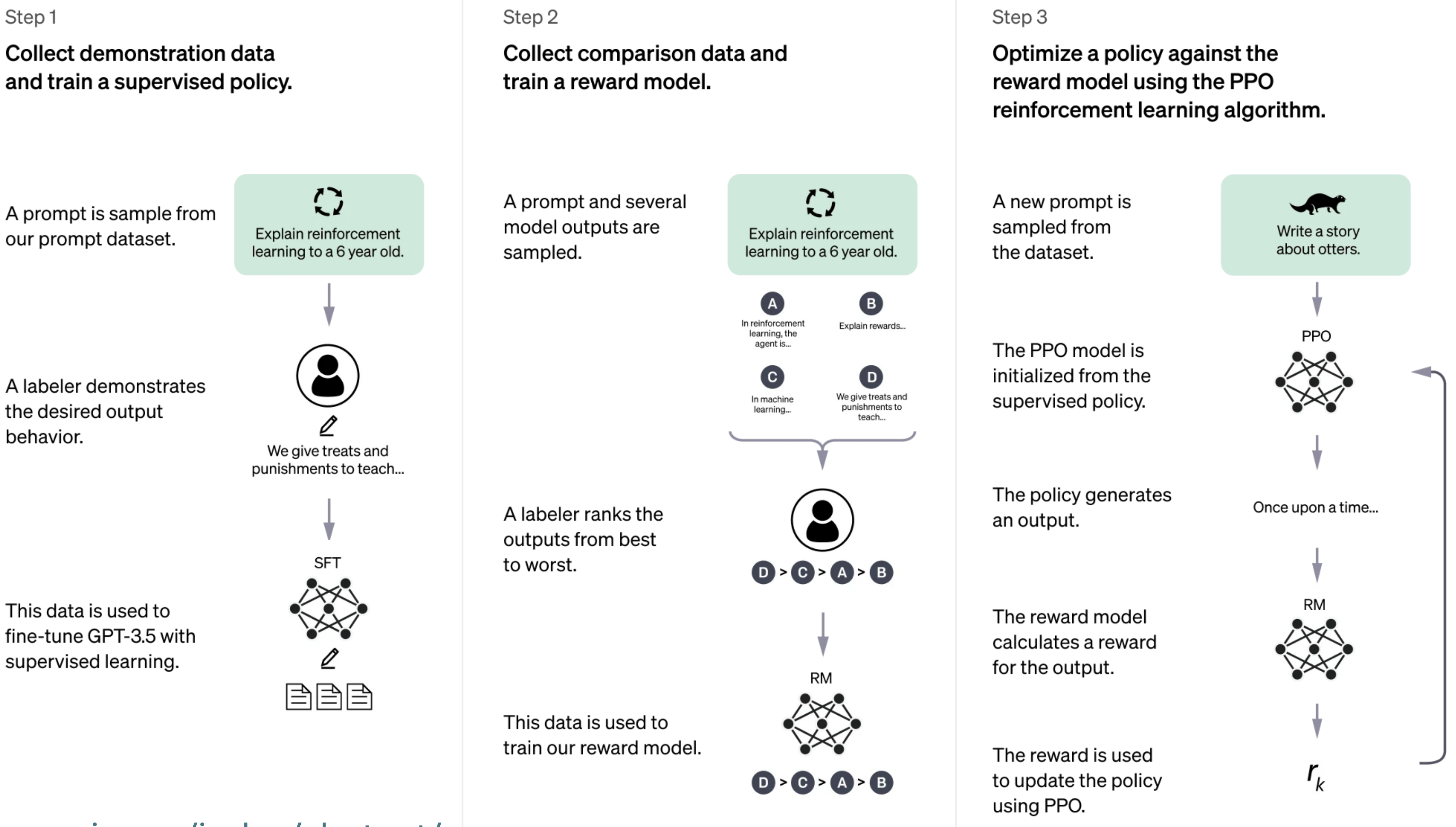
Source: https://huggingface.co/blog/rlhf

RL+Human Feedback (RLHF)



Source: https://huggingface.co/blog/rlhf

Chat-GPT Training Revisited



Source: https://openai.com/index/chatgpt/

DeepSeek

So what did DeepSeek do earlier this year that worked so well?

GRPO: Group Relative Policy Optimization

Sample multiple responses for a given prompt, use relative rewards to train

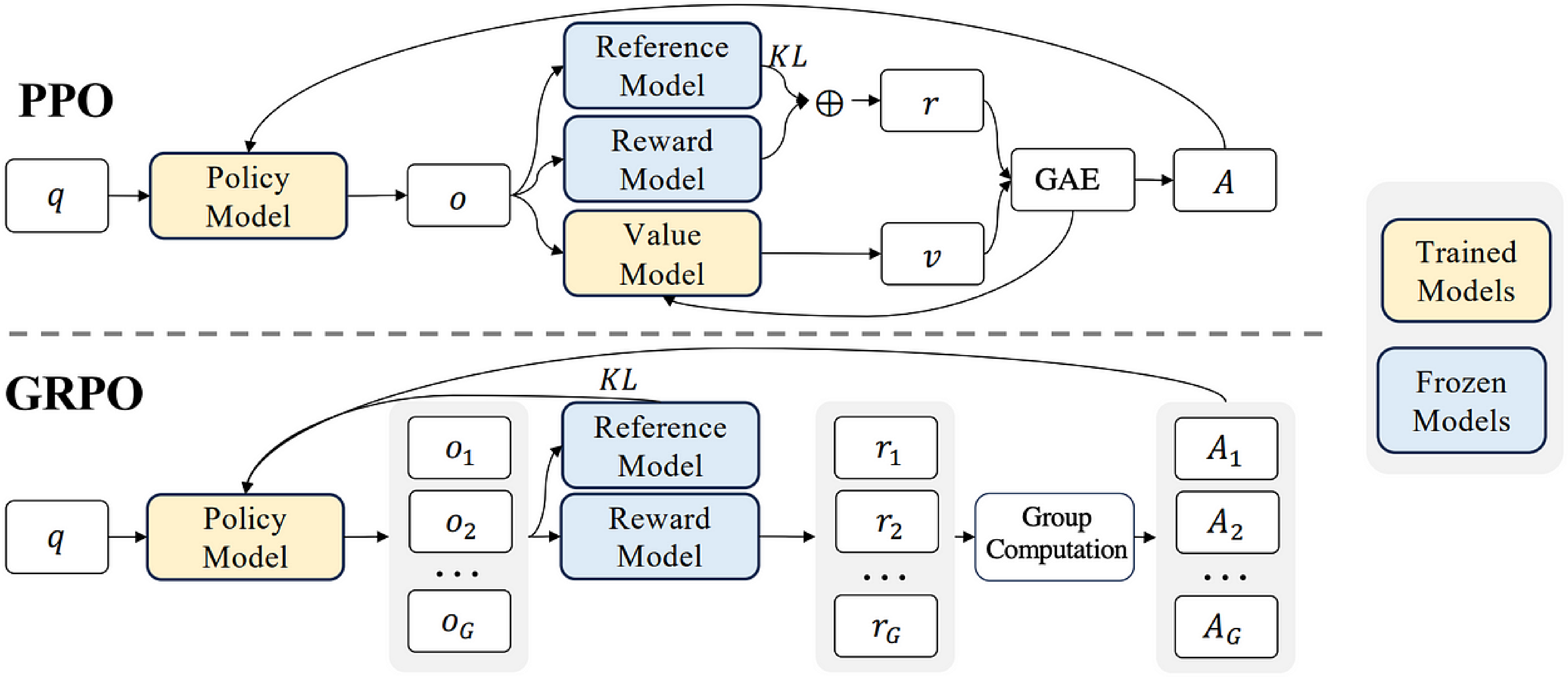
$$\mathcal{J}_{PPO}(\theta) = \mathbb{E}[q \sim P(Q), o \sim \pi_{\theta_{old}}(O|q)] \frac{1}{|o|} \sum_{t=1}^{|o|} \min \left[\frac{\pi_\theta(o_t|q, o_{<t})}{\pi_{\theta_{old}}(o_t|q, o_{<t})} A_t, \text{clip}\left(\frac{\pi_\theta(o_t|q, o_{<t})}{\pi_{\theta_{old}}(o_t|q, o_{<t})}, 1-\varepsilon, 1+\varepsilon \right) A_t \right]$$

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

$$\hat{A}_{i,t} = \widetilde{r}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$$

Don't need a critic model to estimate quality of trajectories, just use normalized rewards for sampled responses for a single prompt. Use KL Divergence directly in loss function.

Source: DeepSeek Math, https://arxiv.org/pdf/2402.03300

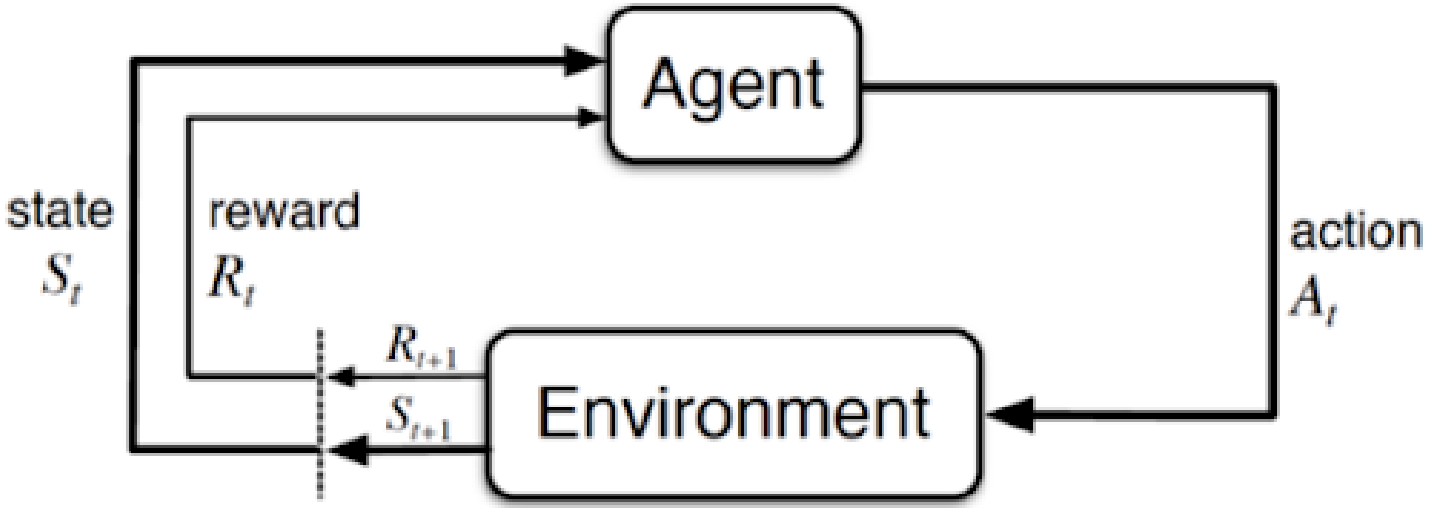


Source: DeepSeek Math, https://arxiv.org/pdf/2402.03300

Robots!

Robots are the most concrete example of autonomous agents

So where are all of the robots trained with RL?



Don’t specify algorithm, but have PPO examples in unitree_rl_gym

Challenges in RL and Robotics

- Simulation environment and real world won't match perfectly (Sim2Real Gap)
  - Hard to collect enough data in the real world
  - Impossible to simulate physics perfectly
- No guarantees of safe policies
  - If you follow a learned and cause an accident, that's very expensive
- Sparse/Delayed rewards
  - It is challenging for a robot to know if it is doing well until a task is complete
- Partial Observability in the real world
  - Robots do not have access to the entire world state, just what they can observe with their sensors.

Why don’t we see more RL in deployed robots?

Why don’t we see more RL in deployed robots?

Deep Learning is not the answer to every problem

We already know optimal-control algorithms for certain types of problems, Deep RL cannot be better than optimal solutions…

But there's lots of problems left

How could we create **generally** intelligent robots?

General Intelligence

What properties do we want from a generally intelligent robot?

1. Adapt to new environments and tasks quickly
2. Goal alignment and value learning
3. Work with multi-modal data
4. Safe exploration and failure recovery
5. Long term memory and experience integration
6. Explainability and Interpretability

Adapt After Training: Continual Learning

What do you do when you encounter new data?

Keep trying to update your model…

2 things may go wrong:

Catastrophic Forgetting: The network no longer knows how to complete a task it once knew

Loss of Plasticity: The network can no longer learn and adapt to new tasks

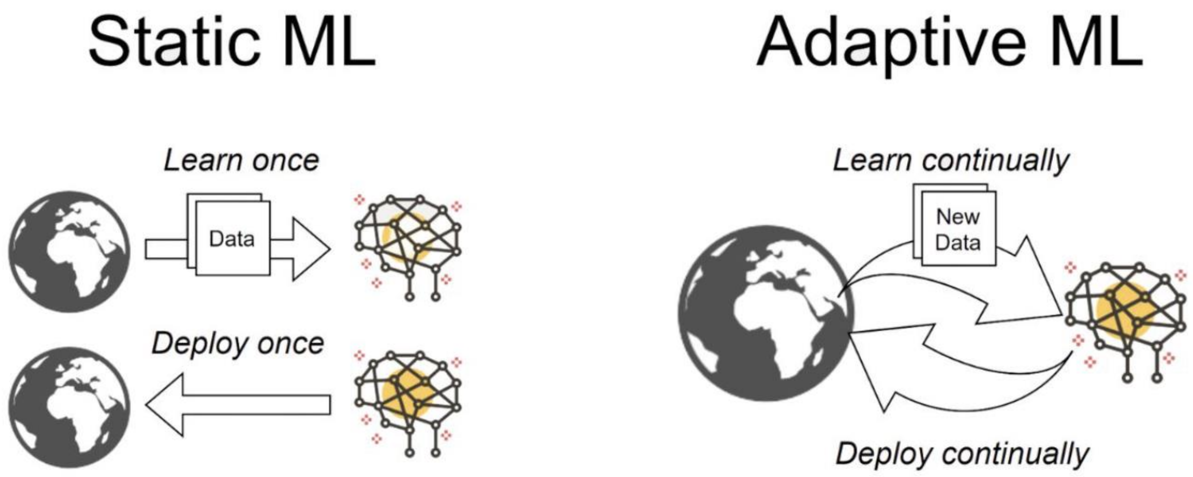
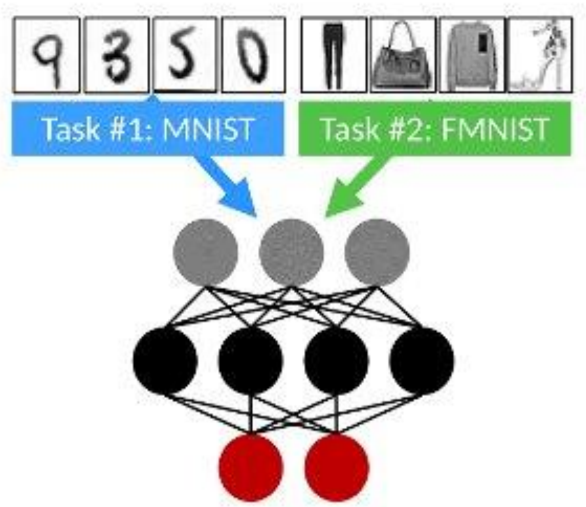


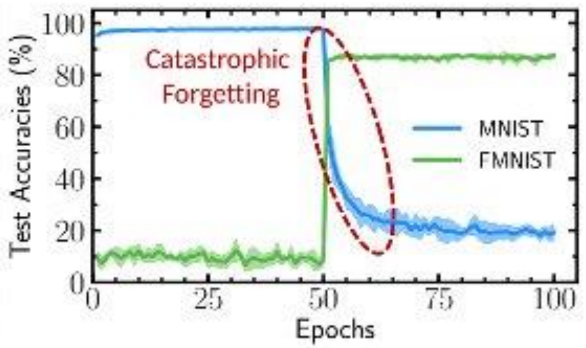
Image source: https://imerit.net/blog/a-complete-introduction-to-continual-learning/

Catastrophic Forgetting

Train network on MNIST, then switch to FMNIST (separate outputs)

Ideally, our networks would remember how to complete the MNIST task



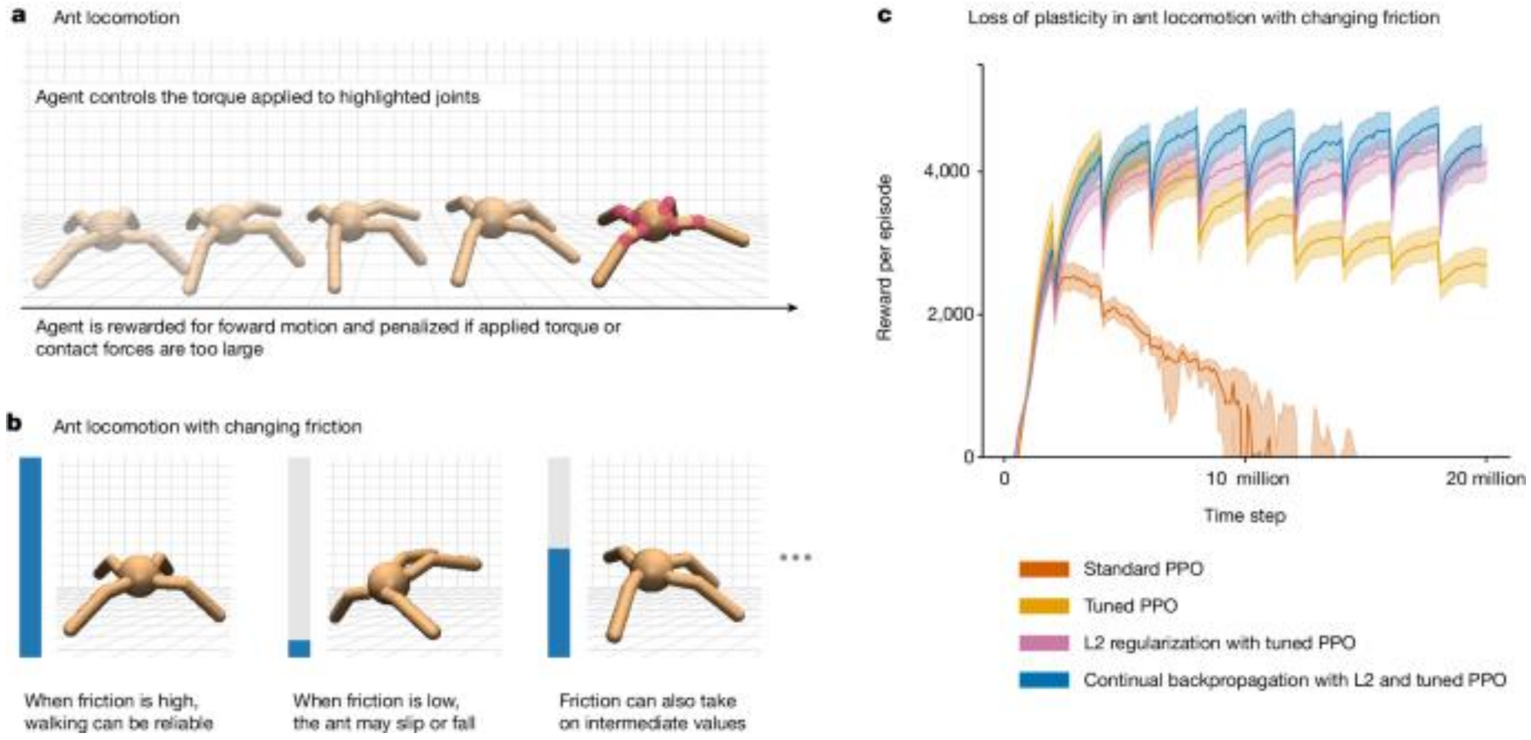


Source: https://www.nature.com/articles/s41467-021-22768-y

Loss Of Plasticity

Catastrophic forgetting is a problem whenever the task switches

But even worse… the network may not learn to complete new tasks



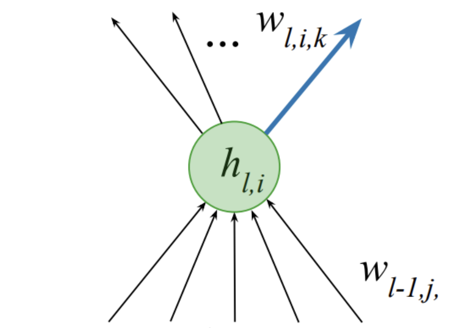
Source: Loss of Plasticity in Continual Deep Learning https://www.nature.com/articles/s41586-024-07711-7

Continual Backprop

Calculate *utility* of each neuron in network

Reinitialize neurons that do not contribute to the output

Continue to run SGD on dataset

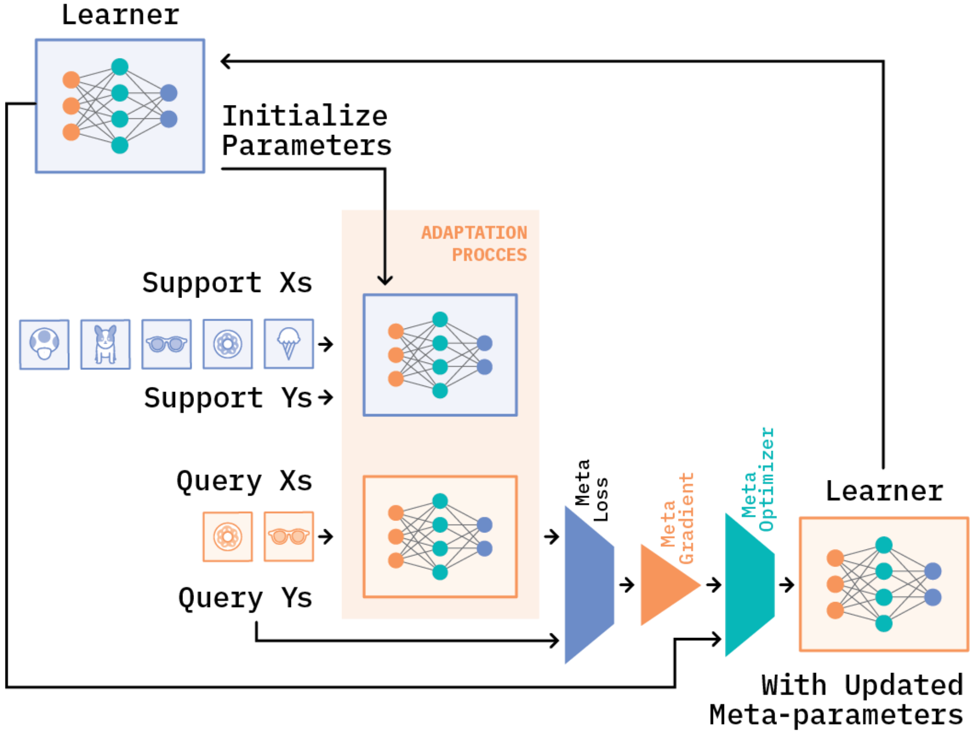


$$y_{l,i,t} = \frac{|h_{l,i,t} - \hat{f}_{l,i,t}| * \sum_{k=1}^{n_{l+1}} |w_{l,i,k,t}|}{\sum_{j=1}^{n_{l-1}} |w_{l-1,j,i,t}|}$$

Figure 4: A feature/hidden-unit in a network. The utility of a feature at time t is the product of its contribution utility and its adaptation utility. Adaptation utility is the inverse of the sum of the magnitude of the incoming weights. And, contribution utility is the product of the magnitude of the outgoing weights and feature activation ($h_{l,i}$) minus its average ($\hat{f}_{l,i}$). $\hat{f}_{l,i}$ is a running average of $h_{l,i}$.

Adapting to New Tasks: Meta-Learning

Train a model that can adapt **quickly** to new tasks



Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

randomly initialize θ
while not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  for all $\mathcal{T}_i$ **do**
    Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
    Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
  end for Note: the meta-update is using different set of data.
  Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
end while

Model Agnostic Meta-Learning (MAML)

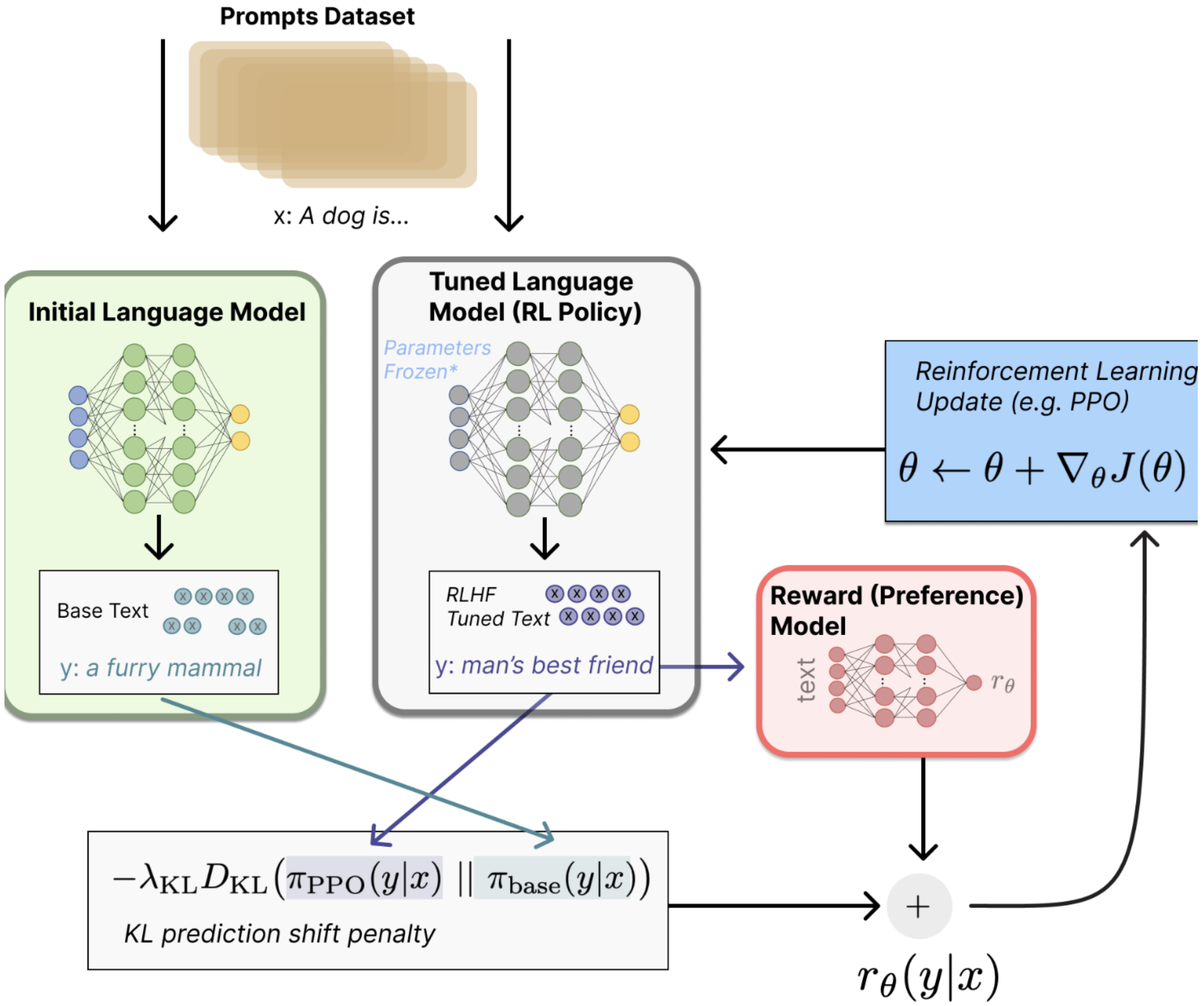
ource: https://medium.com/aimonks/mastering-adaptability-exploring-model-agnostic-meta-learning-maml-996796c890f0

General Intelligence

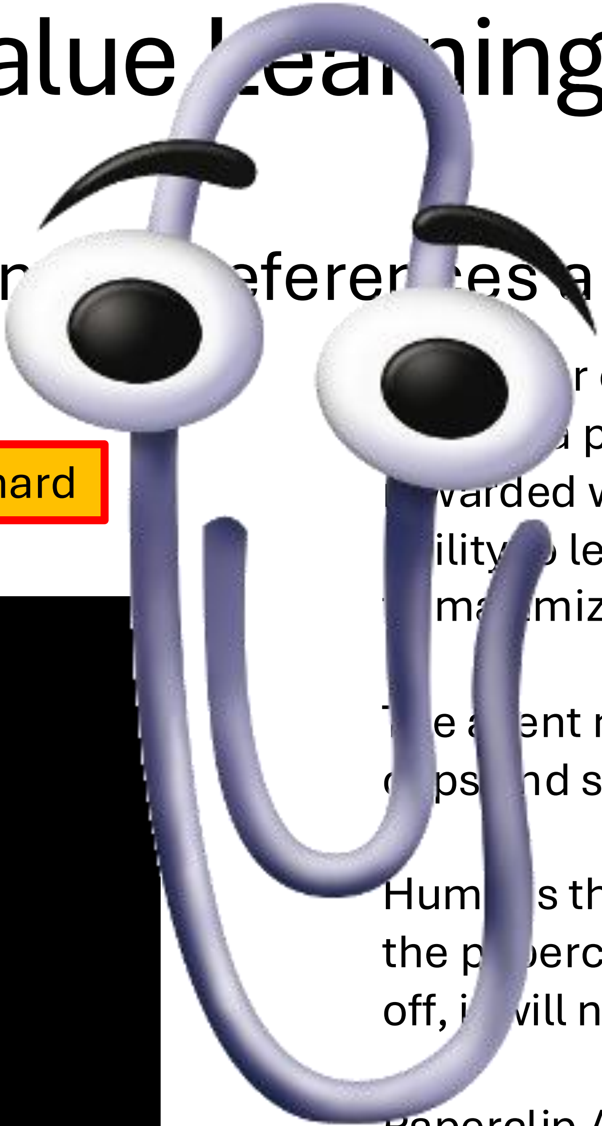
What properties do we want from a generally intelligent robot?

1. Adapt to new environments and tasks quickly
2. **Goal alignment and value learning**
3. Work with multi-modal data
4. Safe exploration and failure recovery
5. Long term memory and experience integration
6. Explainability and Interpretability

RLHF is a way to perform alignment



Alignment and Value L ing



How can robots learn hu eferen es and what we want them to do?

Specifying reward functions is hard

Positive reward for surviving, negative reward for losing

r clip parable:
paper clip factory and train an agent that is
varded when it produces a paper clip. We give it the
ility learn even better strategies. The agent wants
ma mize reward.

e ent needs to secure more resources for paper
ps nd starts strip mining.

Hum s think strip mining is bad, and want to turn off the p perclip AI. The paperclip AI knows if it is turned off, i vill no longer get rewards.

Paperclip AI wipes out humanity so that it can continue to make paperclips.

General Intelligence

What properties do we want from a generally intelligent robot?

1. Adapt to new environments and tasks quickly
2. Goal alignment and value learning
3. **Work with multi-modal data**
4. Safe exploration and failure recovery
5. Long term memory and experience integration
6. Explainability and Interpretability

Working with Multi-Modal Data

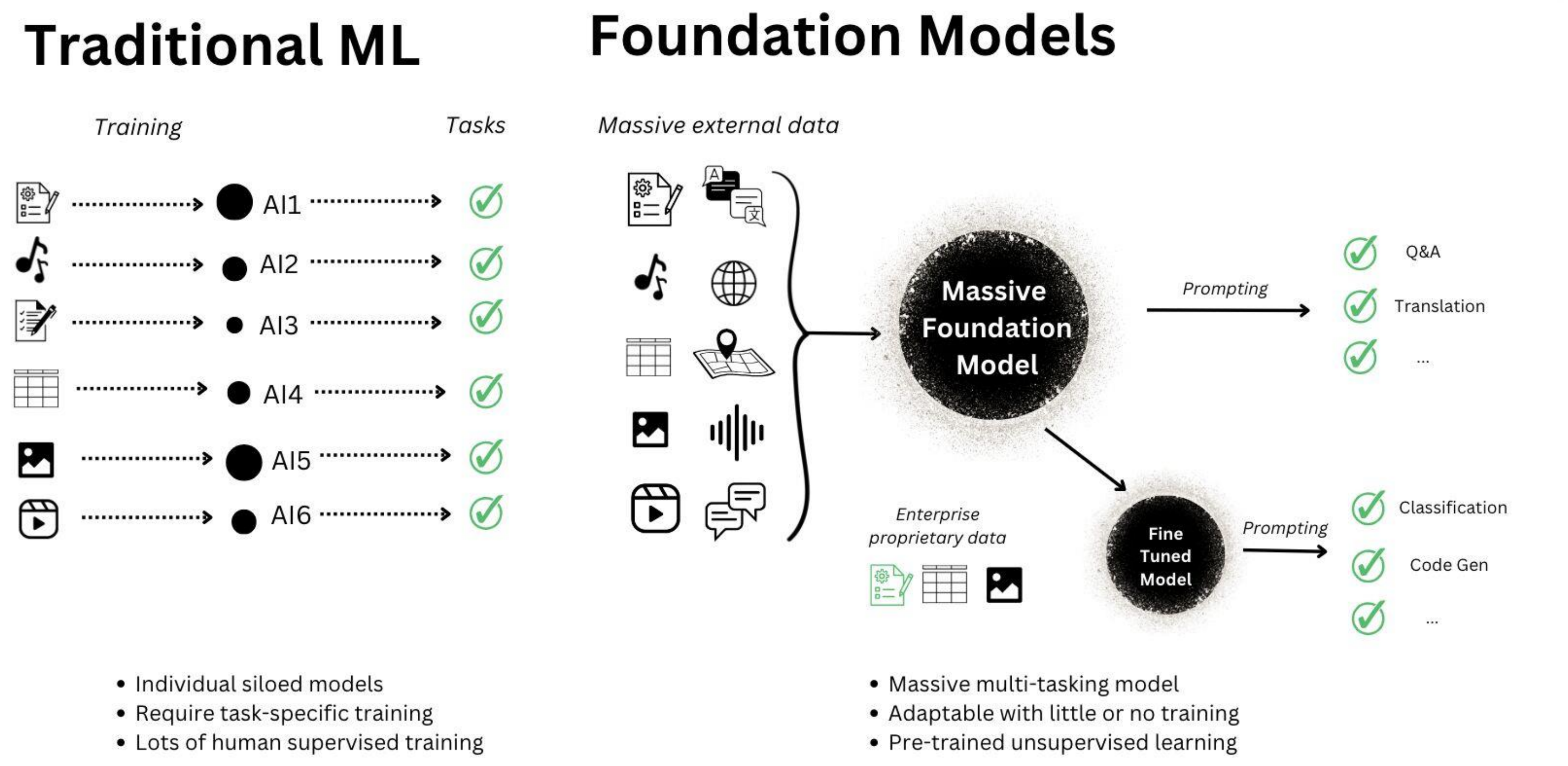


Image source: https://humanloop.com/blog/foundation-models