Filippos Christianos

Real-world
Multi-Agent Systems:
Research and Applications

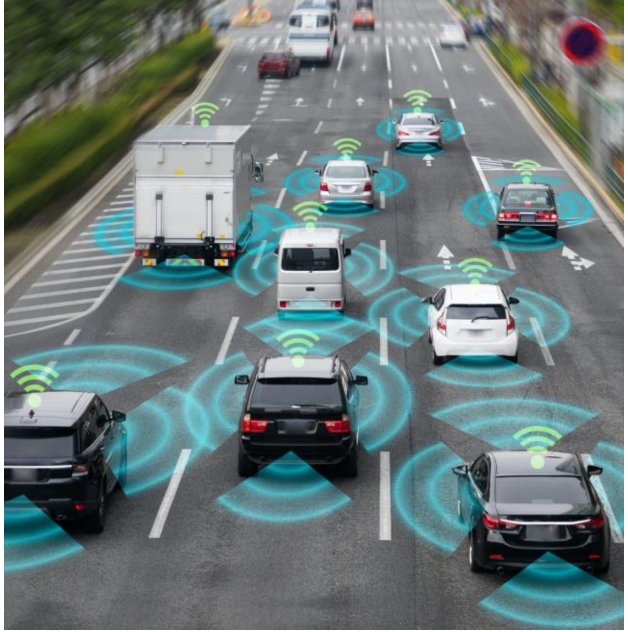
About me

- Research Scientist at Huawei
- PhD in the University of Edinburgh
- Research Scientist, Intern at Nvidia
- Electrical and Computer Engineering

- Co-Author of the textbook "**Multi-Agent Reinforcement Learning**"
  www.marl-book.com (MIT Press)

In the next two hours…

Part I	Introduction, Definitions, History of MARL
Part II	Robotic Warehouses & Collaborative Environments: Scaling to Many Agents
Part III	MARL in Practice
Part IV	AI Agents: Introduction
Part V	LLMs for Mobile-Phone Control

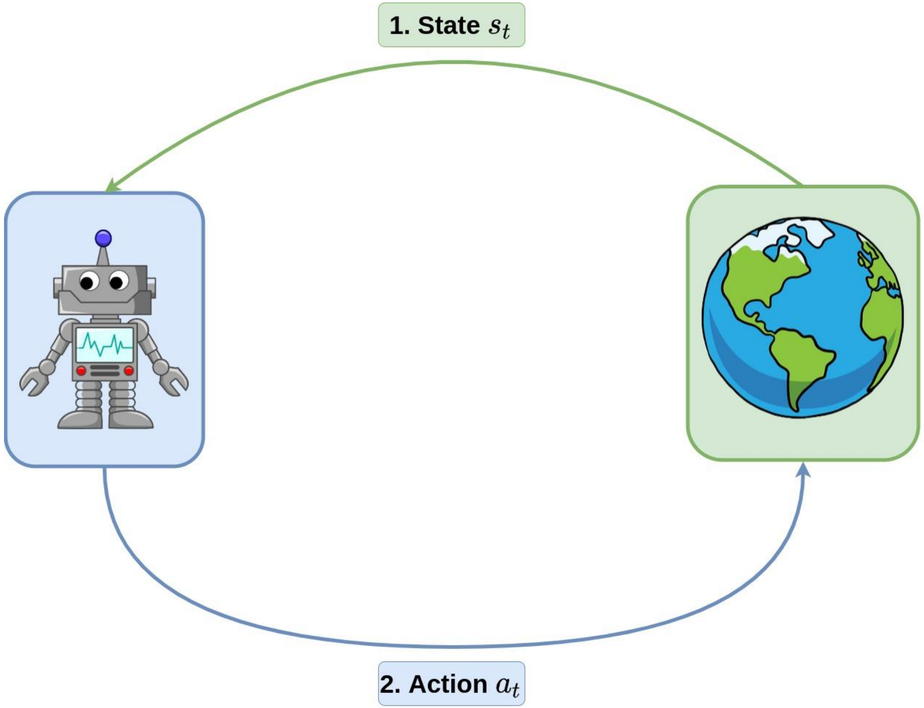


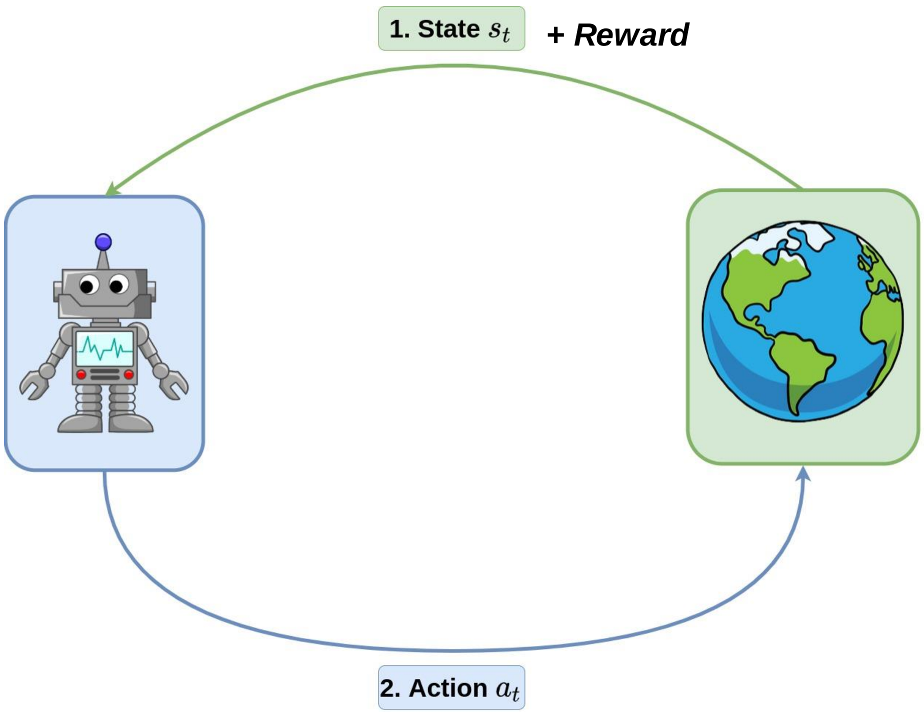
Research Goal:

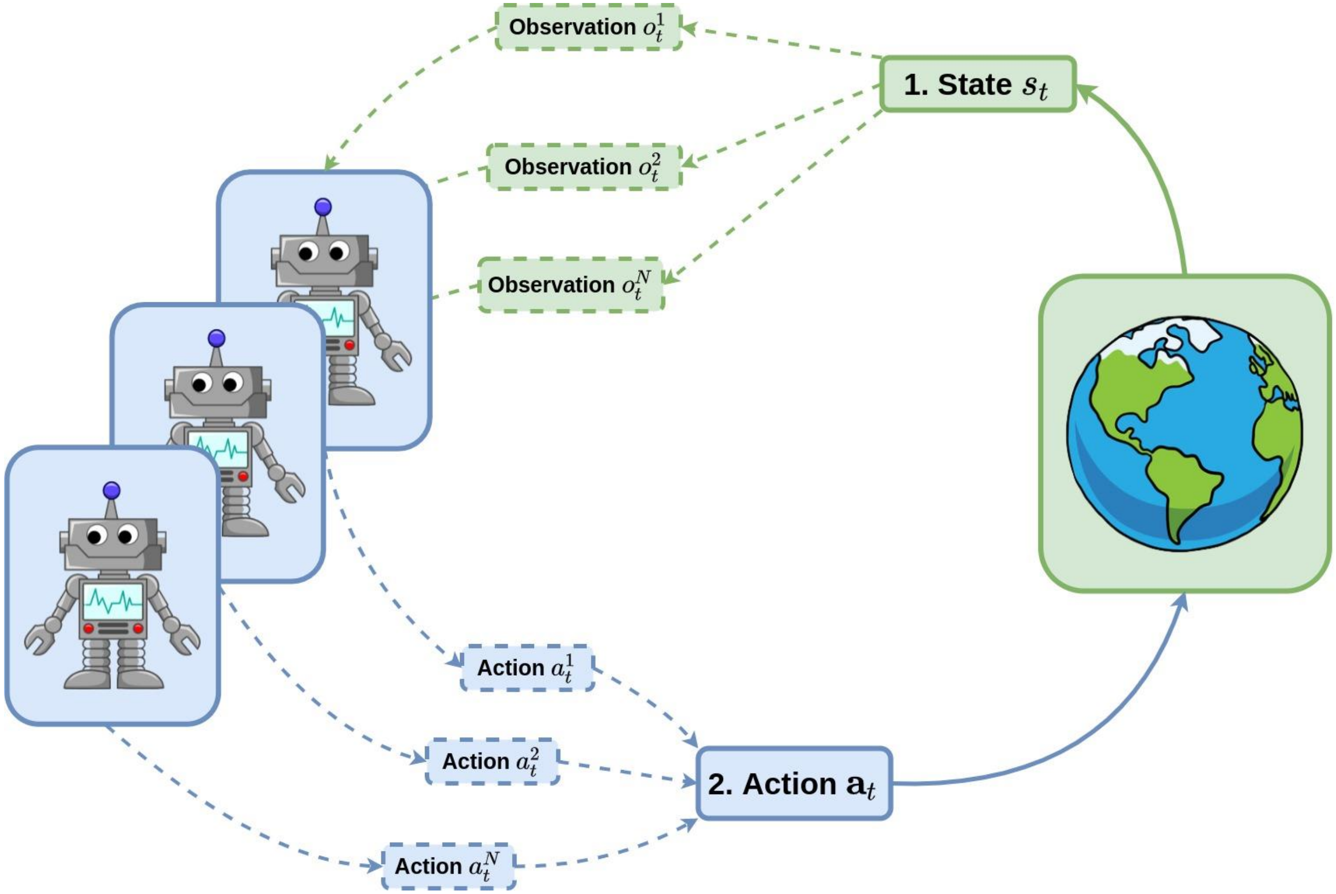
Create autonomous agents which can accomplish tasks in complex dynamic environments

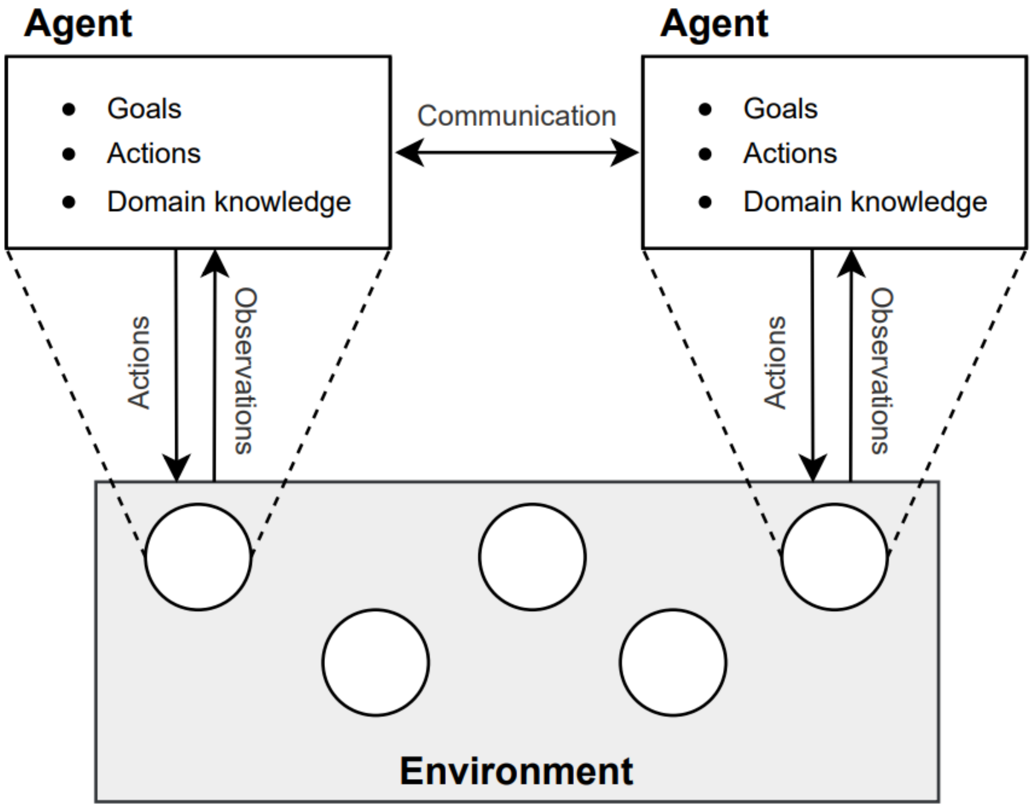
Part I: Multi-Agent Systems – Scope and Definitions

1. What is an agent?
2. What is a multi-agent environment?
3. How do multiple agents *learn*?
4. Historical background and current state of research


1. State s_t
2. Action a_t


1. State s_t
+ *Reward*
2. Action a_t


Observation o_t^1
Observation o_t^2
Observation o_t^N
1. State s_t
Action a_t^1
Action a_t^2
Action a_t^N
2. Action $\mathbf{a}_t$


Agent
Goals
Actions
Domain knowledge
Communication
Agent
Goals
Actions
Domain knowledge
Actions
Observations
Actions
Observations
Environment

How did it start?

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

Rock Paper Scissors (RPS)

	A	B
A	10	0
B	0	10

Coordination Game

	C	D
C	-1,-1	-5,0
D	0,-5	-3,-3

Prisoner’s Dilemma

Solution Concepts

Best Response

Nash Equilibrium

Minimax

Pareto Optimality

…No-Regret, Social Welfare, Fairness… *and more.*

Solution Concepts

Best Response

Nash Equilibrium

Minimax

Pareto Optimality

…No-Regret, Social Welfare, Fairness… *and more.*

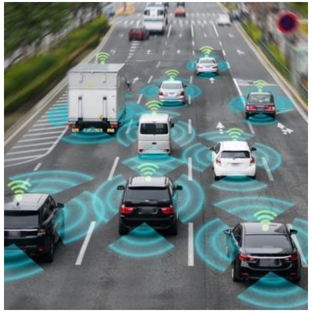
	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

Rock Paper Scissors (RPS)

How did we get where we are now?

	R	P	S
R	0,0	-1,1	1,-1
P	1,-1	0,0	-1,1
S	-1,1	1,-1	0,0

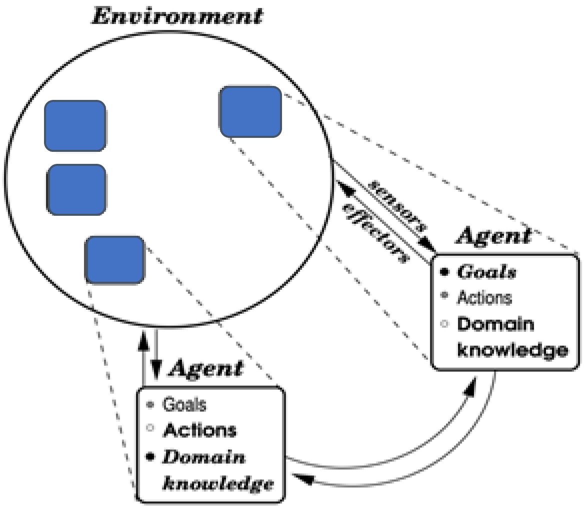
Rock Paper Scissors (RPS)



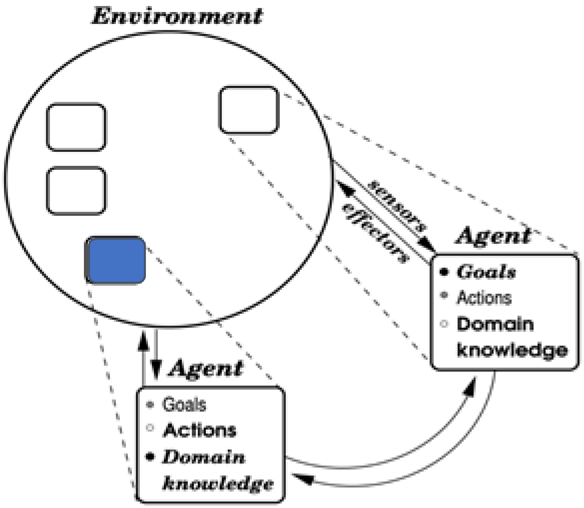
?

Two Types of Multi-agent Learning

We control all agents:



We control one agent:



Two Types of Multi-agent Learning

We control all agents:

How can agents learn to interact optimally?

- *Coordination and communication*
- *Efficient scaling to many agents*
- *Transfer abilities to other agents*

We control one agent:

How can agent learn to collaborate with unknown other agents?

- *Recognising goals of other agents*
- *Integrating prediction and planning*

Part II: Collaborative Training of Multiple Autonomous Agents

How can we train agents in large environments, and ensure they learn useful **joint policies**?

What is the problem here?

- Many Agents!
  - The joint action space is huge
  - … This makes the joint policy search space intractable

Can we do something?

What is the problem here?

Can we do something?

Restricting the policy search space with parameter sharing

What is the problem here?

Can we do something?

Restricting the policy search space with parameter sharing

Joint policy is constrained to consist of identical individual policies!

Parameter Sharing

No Parameter Sharing

Each agent has its own policy

- distinct behaviors can be learned
- agents only learn from their own exploration number of parameters

Parameter Sharing

All agents share the same policy

- sample efficiency (agents can learn from other’s exploration)
- computationally cheap (less parameters)
- learn only homogeneous behaviors

Christianos et al. – Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning, NeurIPS 2020

Parameter Sharing

No Parameter Sharing

Each agent has its own policy

- distinct behaviors can be learned
- agents only learn from their own exploration number of parameters

Shared Experience Actor Critic

Each agent has its own policy – but experience is shared

- distinct behaviors can be learned
- sample efficiency (agents can learn from other's exploration)
- number of parameters

Parameter Sharing

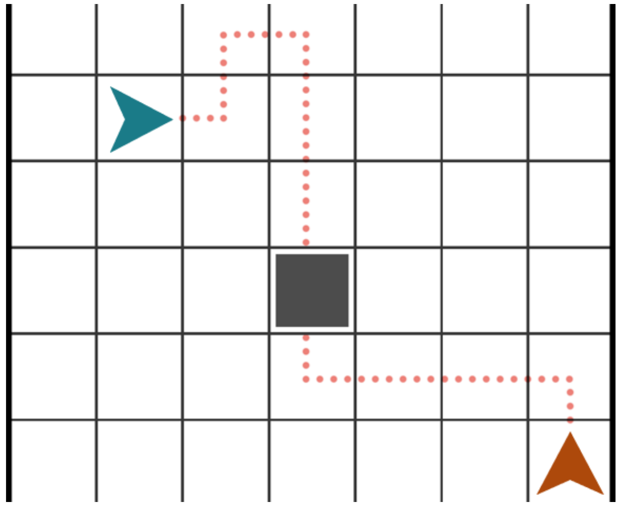
All agents share the same policy

- sample efficiency (agents can learn from other's exploration)
- computationally cheap (less parameters)
- learn only homogeneous behaviors

Christianos et al. – Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning, NeurIPS 2020

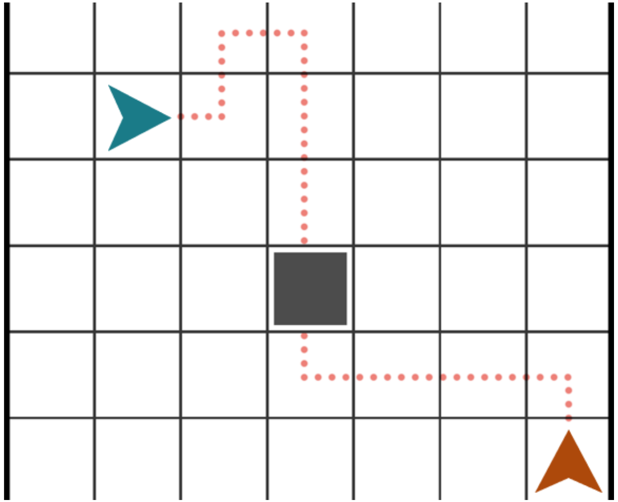
Motivational Example

- Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning



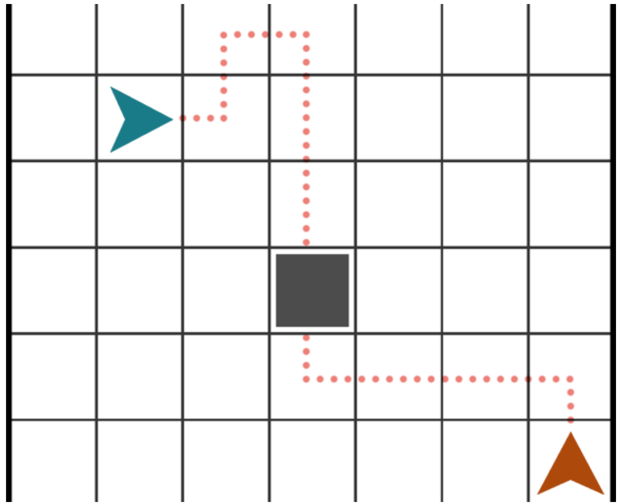
Motivational Example

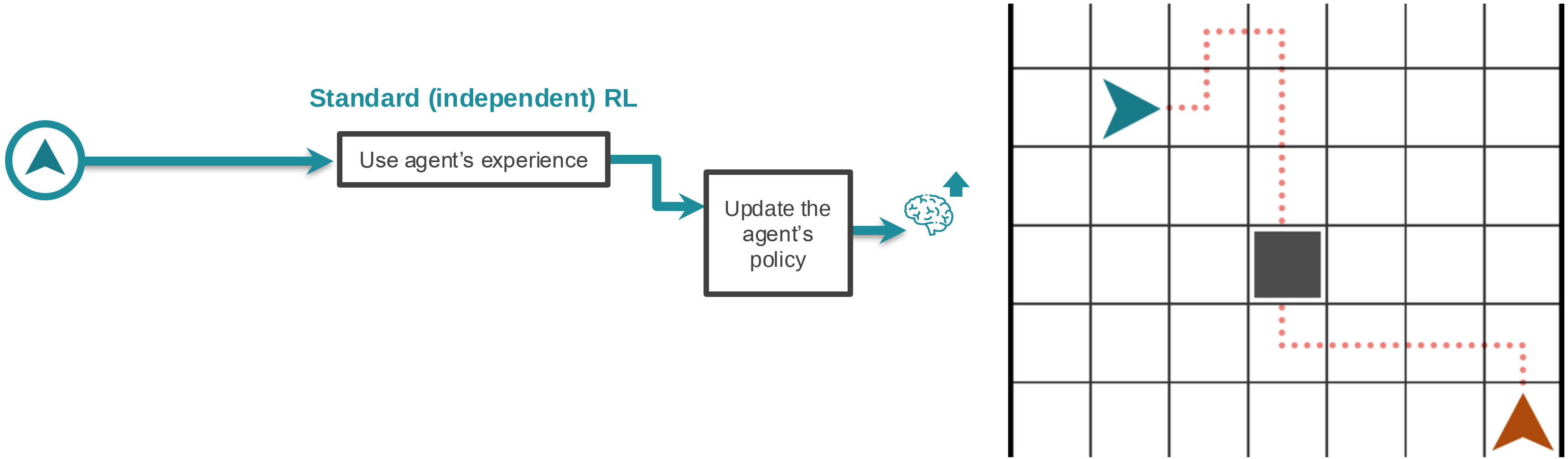
- Both agents must reach goal simultaneously
- Sparse reward signal

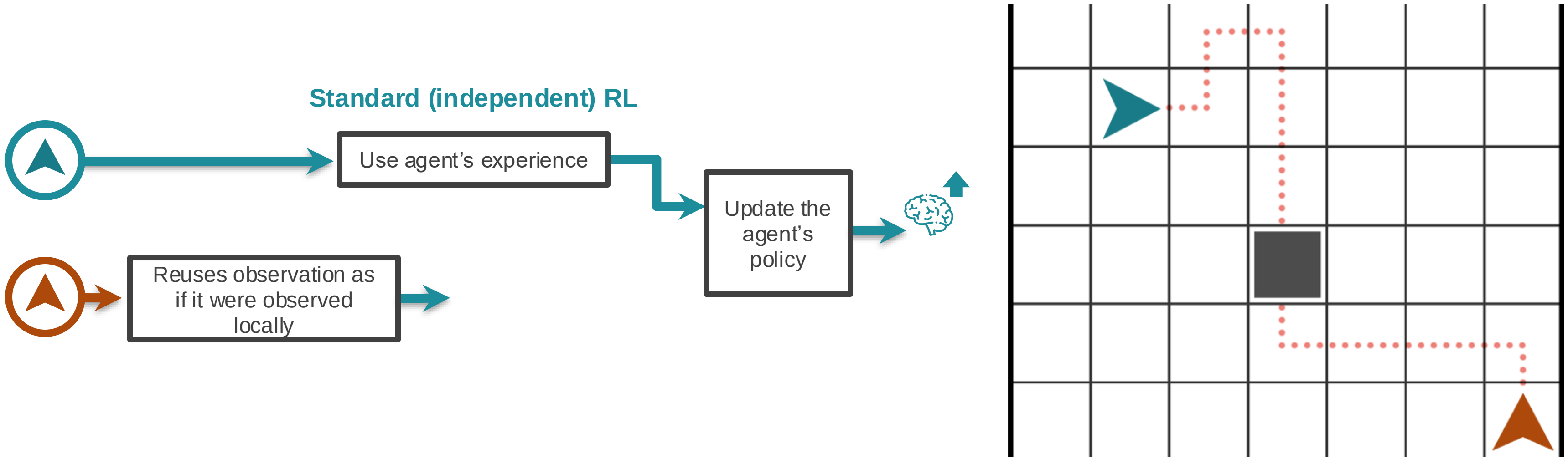


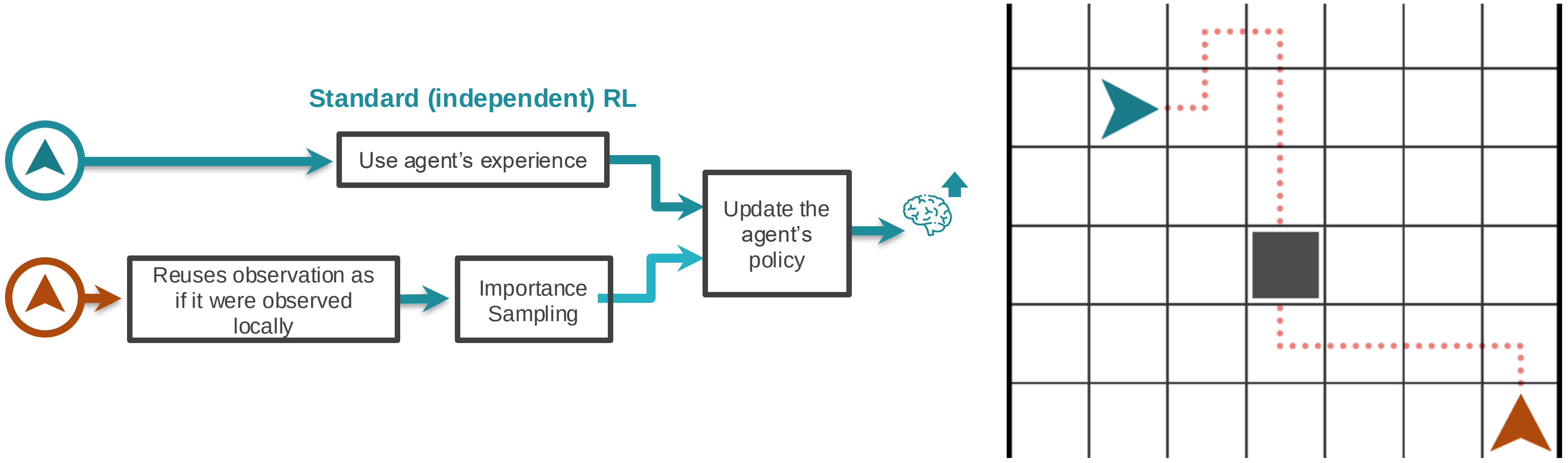
Motivational Example

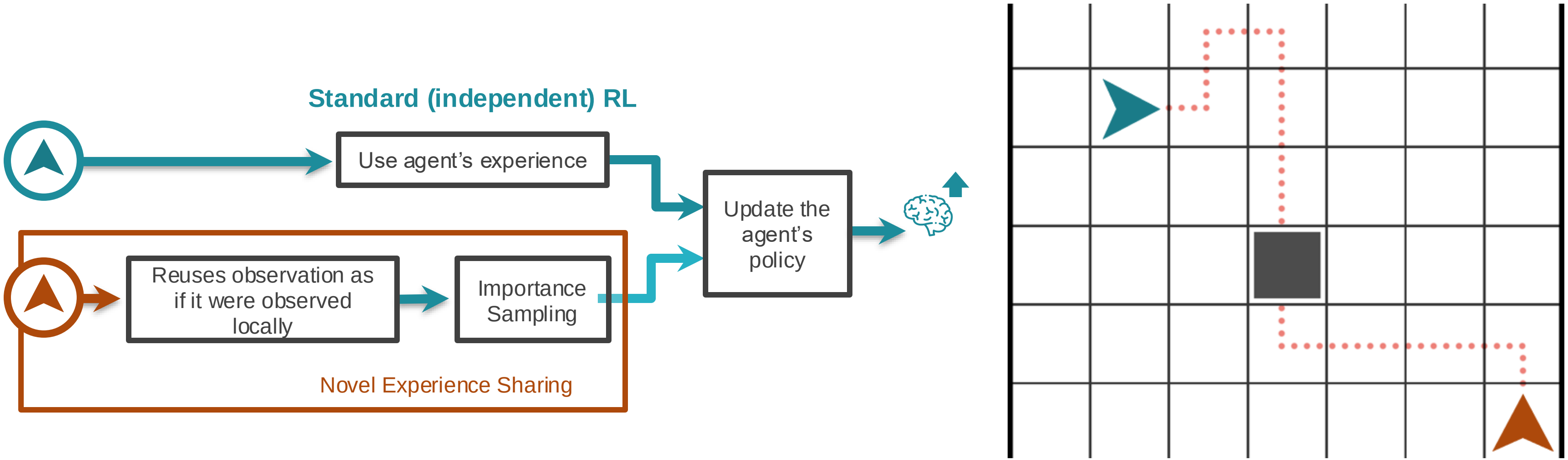
- Both agents must reach goal simultaneously
  - Sparse reward signal
- Idea: Make use of both agents' exploration
  - Share experience of agents




Standard (independent) RL
Use agent’s experience
Update the agent’s policy

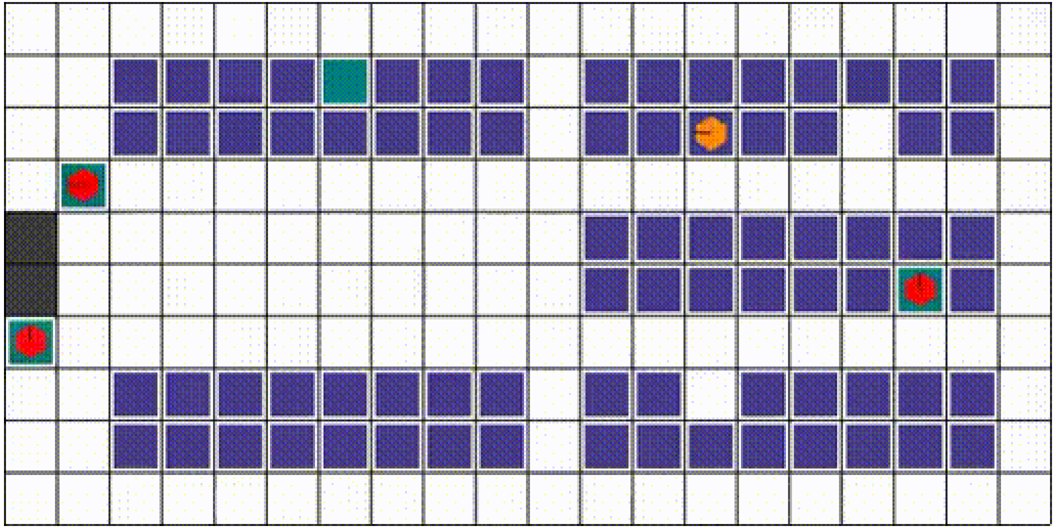

Standard (independent) RL
Use agent’s experience
Update the agent’s policy
Reuses observation as if it were observed locally

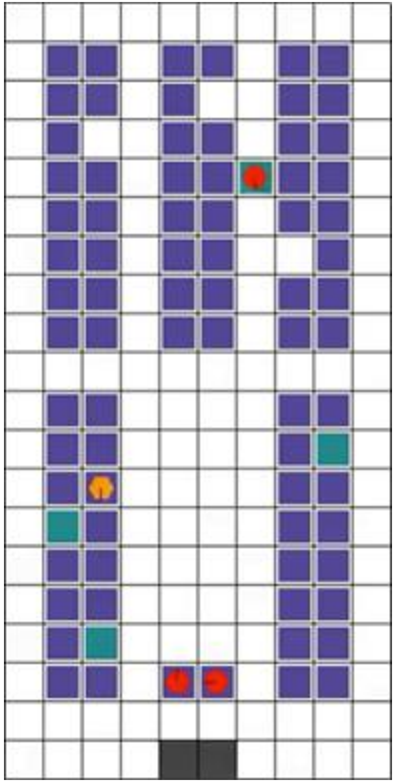

Standard (independent) RL
Use agent's experience
Reuses observation as if it were observed locally
Importance Sampling
Update the agent's policy

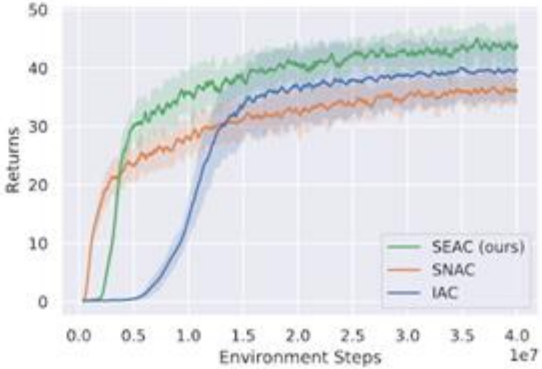


Christianos et al. 2021. Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning

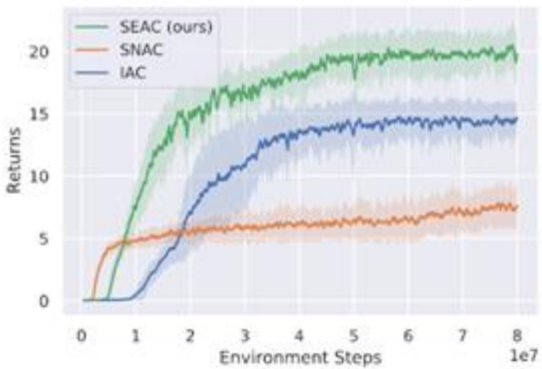
A simulated robotics warehouse



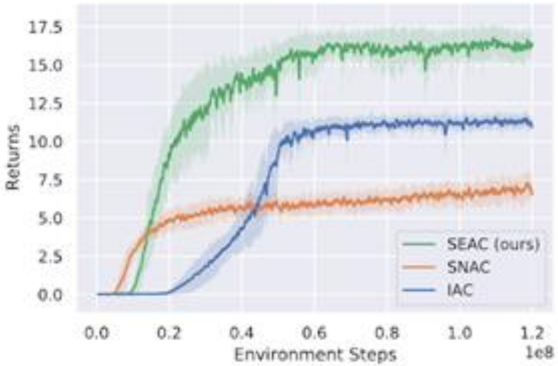




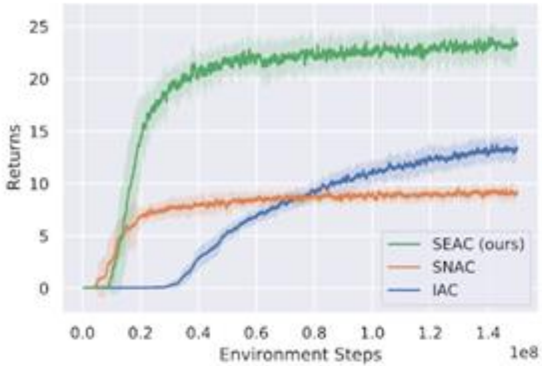
(a) RWARE: (10 × 11), four agents



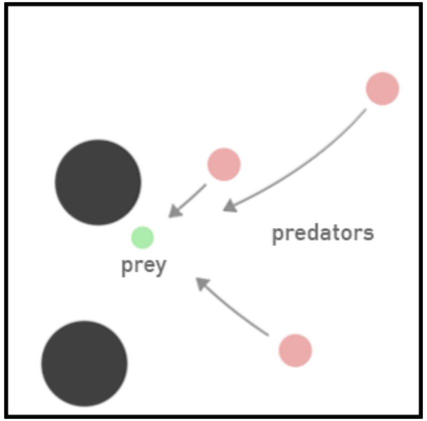
(b) RWARE: (10 × 11), two agents



(c) RWARE: (10 × 11), two agents, hard

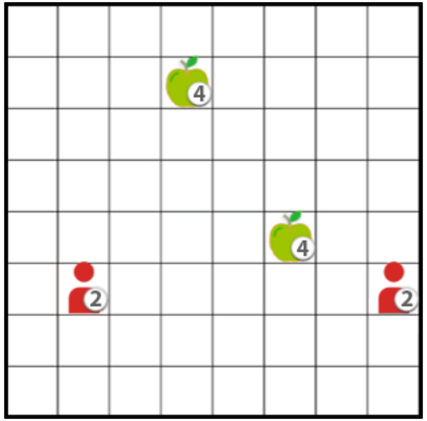


(d) RWARE: (10 × 20), four agents

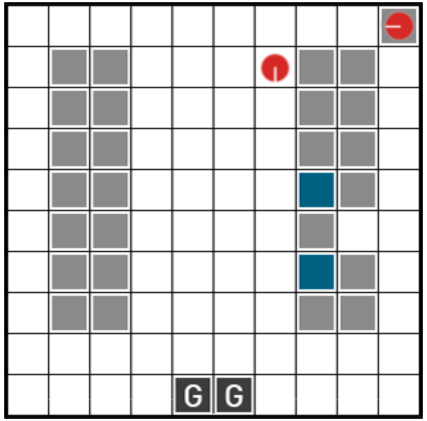


Predator Prey (sparse)

SMAC – 3m (sparse)



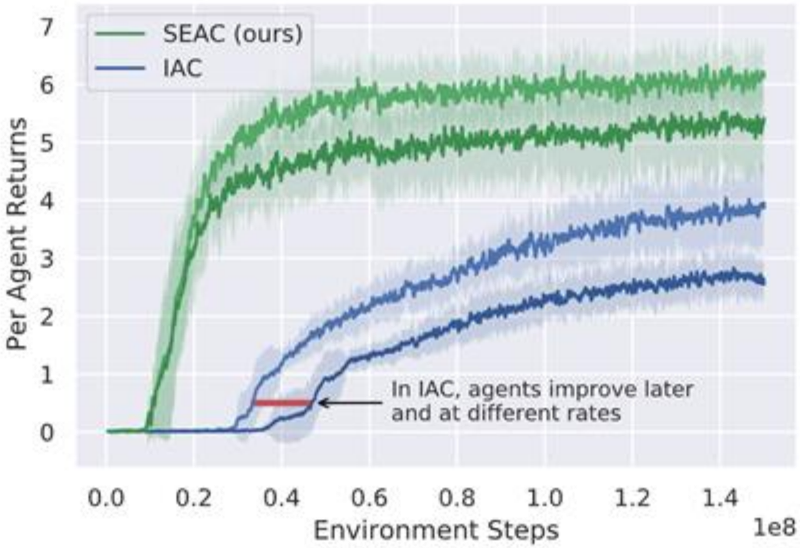
Level-Based Foraging (LBF)



Multi-Robot Warehouse (RWARE)

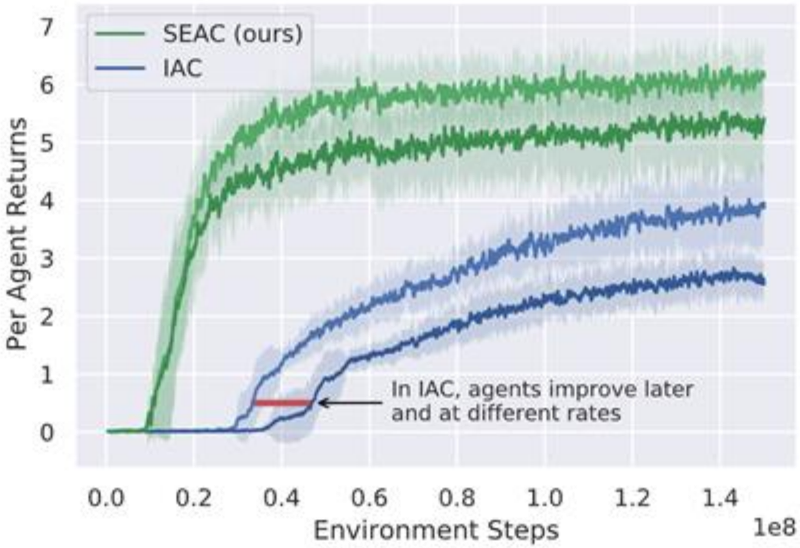
	IAC	SNAC	SEAC (ours)	QMIX	MADDPG	ROMA
PP (sparse)	-0.04 ±0.13	-0.04 ±0.1	**1.93 ±0.13**	0.05 ±0.07	**2.04 ±0.08**	0.04 ±0.07
SMAC-3m (sparse)	-0.13 ±0.01	-0.14 ±0.02	**-0.03 ±0.03**	**0.00 ±0.00**	**-0.01 ±0.01**	**0.00 ±0.00**
LBF-(15x15)-3ag-4f	0.13 ±0.04	0.18 ±0.08	**0.43 ±0.09**	0.03 ±0.01	0.01 ±0.02	0.03 ±0.02
LBF-(8x8)-2ag-2f-coop	0.37 ±0.10	0.38 ±0.10	**0.64 ±0.08**	**0.79 ±0.31**	0.01 ±0.02	0.01 ±0.02
RWARE-(10x20)-4ag	13.75 ±1.26	9.53 ±0.83	**23.96 ±1.92**	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00
RWARE-(10x11)-4ag	**40.10 ±5.60**	36.79 ±2.36	**45.11 ±2.90**	0.00 ±0.00	0.00 ±0.00	0.01 ±0.01

Analysis



Best vs. Worst performing agents on RWARE, (10x20), four agents

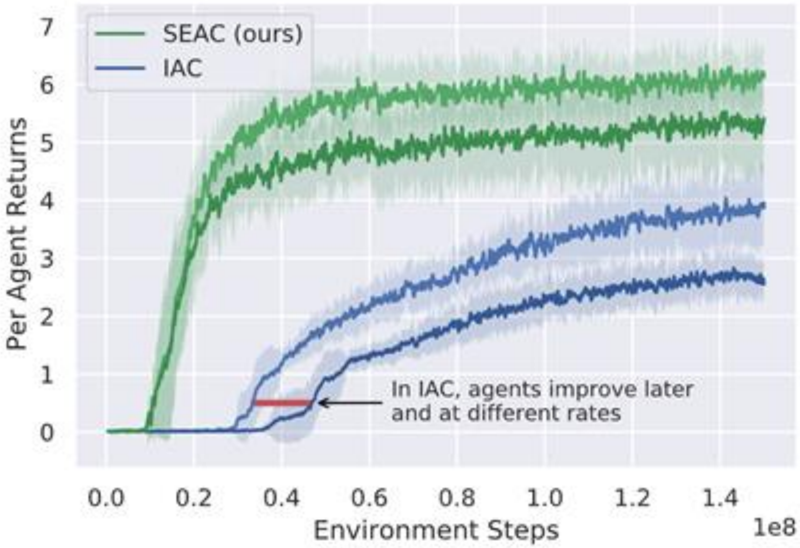
Analysis



Best vs. Worst performing agents on RWARE, (10x20), four agents

- Agents learn simultaneously which helps in exploring promising joint actions more

Analysis



Best vs. Worst performing agents on RWARE, (10x20), four agents

- Agents learn simultaneously which helps in exploring promising joint actions more
- Synchronise training progress of agents

Environments with heterogeneous agents

No Parameter Sharing

Each agent has its own policy

- distinct behaviors can be learned
- agents only learn from their own exploration number of parameters

Shared Experience Actor Critic

Each agent has its own policy – but experience is shared

- distinct behaviors can be learned
- sample efficiency (agents can learn from other's exploration)
- number of parameters

Parameter Sharing

All agents share the same policy

- sample efficiency (agents can learn from other's exploration)
- computationally cheap (less parameters)
- learn only homogeneous behaviors

Selective Parameter Sharing

Groups of agents that share policies

- distinct behaviors can be learned
- sample efficiency (agents can learn from other’s exploration)
- computationally cheap

No Parameter Sharing

Each agent has its own policy

- distinct behaviors can be learned
- agents only learn from their own exploration number of parameters

Shared Experience Actor Critic

Each agent has its own policy – but experience is shared

- distinct behaviors can be learned
- sample efficiency (agents can learn from other’s exploration)
- number of parameters

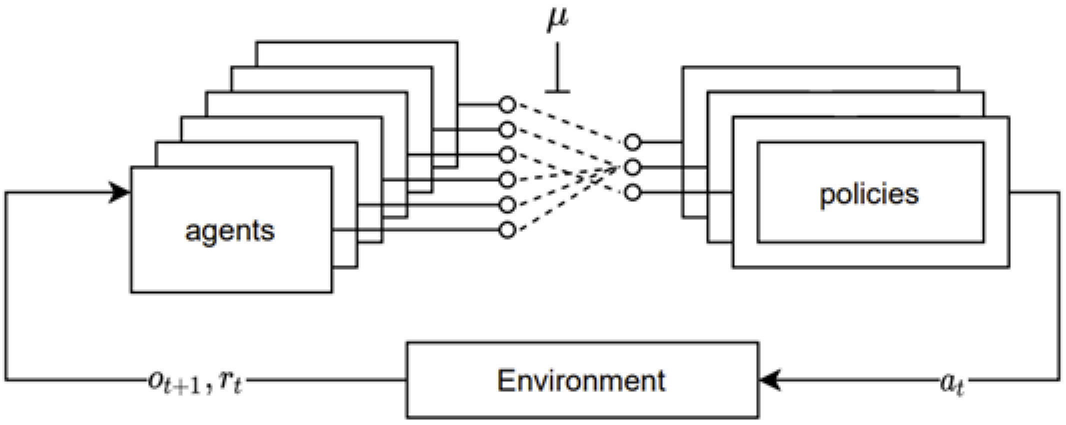
Parameter Sharing

All agents share the same policy

- sample efficiency (agents can learn from other’s exploration)
- computationally cheap (less parameters)
- learn only homogeneous behaviors

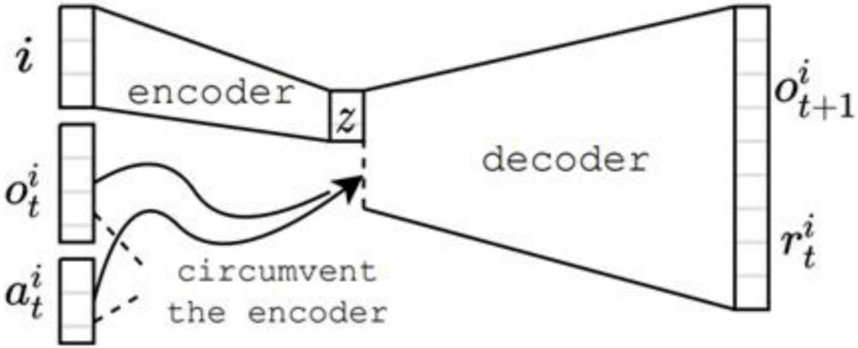
Selective Parameter Sharing

But we can apply it selectively.

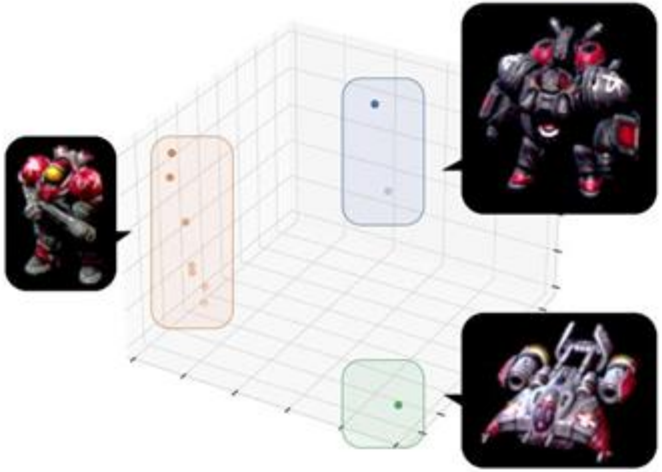


Selective Parameter Sharing

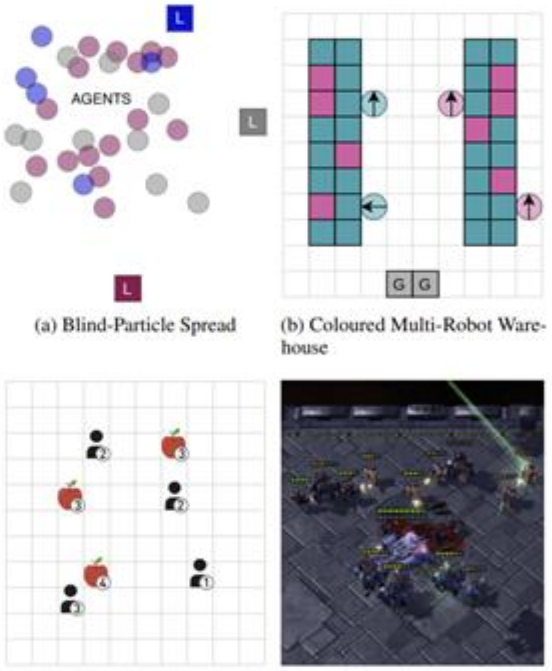
We identify agents with similar reward and observation transition functions and have them share parameters.



Visualising the embedding space



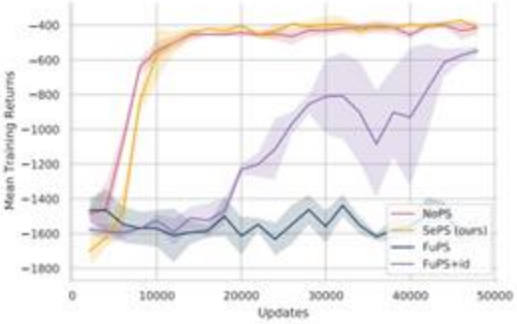
Experiments: Environments



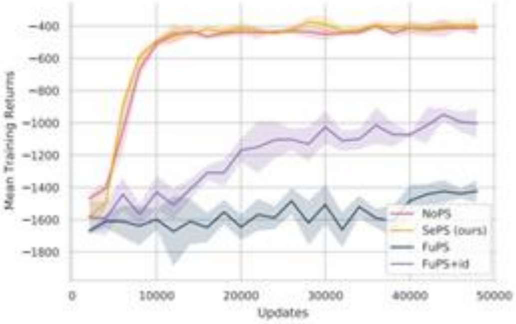
(a) Blind-Particle Spread (b) Coloured Multi-Robot Warehouse

(c) Level-based Foraging (d) SMAC

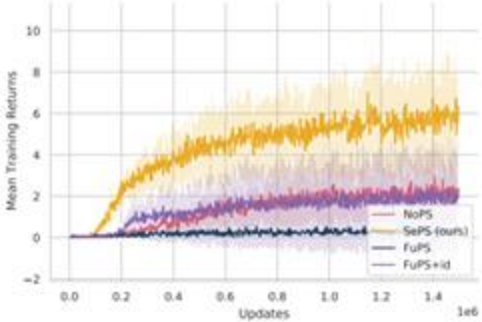
	# Agents	# Types	Type Distribution
BPS (1)	15	3	5–5–5
BPS (2)	30	3	10–10–10
BPS (3)	30	5	6–6–6–6–6
BPS (4)	30	5	2–2–2–15–9
BPS-h (1)	15	$3^{\dagger}$	5–5–5
BPS-h (2)	30	$5^{\dagger}$	6–6–6–6–6
BPS-h (3)	200	$4^{\dagger}$	50–50–50–50
C-RWARE (1)	4	$2^{\ddagger}$	2–2
C-RWARE (2)	8	$2^{\ddagger}$	4–4
C-RWARE (3)	16	$2^{\ddagger}$	8–8
LBF	12	3	4–4–4–4
MMM2	10	$3^{\S}$	7–2–1



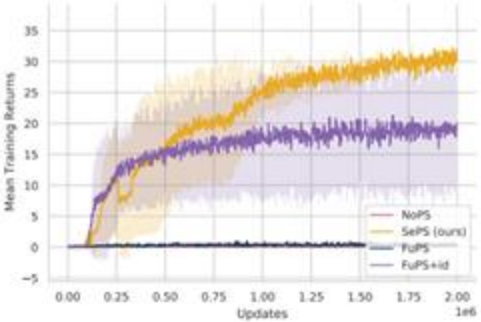
(a) BPS (3)



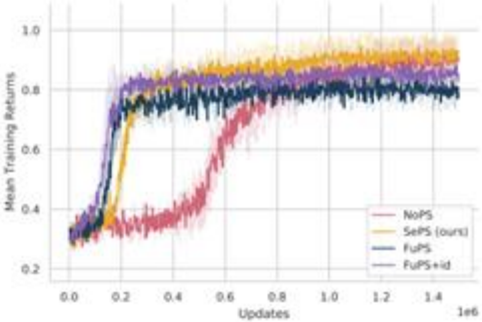
(b) BPS-h (2)



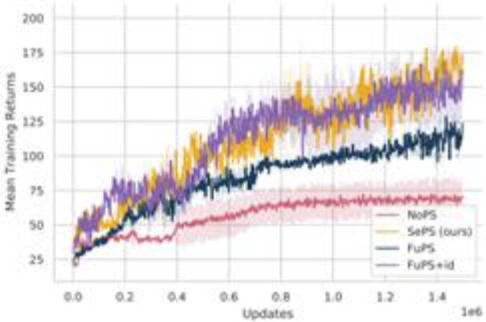
(c) C-RWARE (1)



(d) C-RWARE (3)



(e) LBF



(f) SMAC (MMM2)

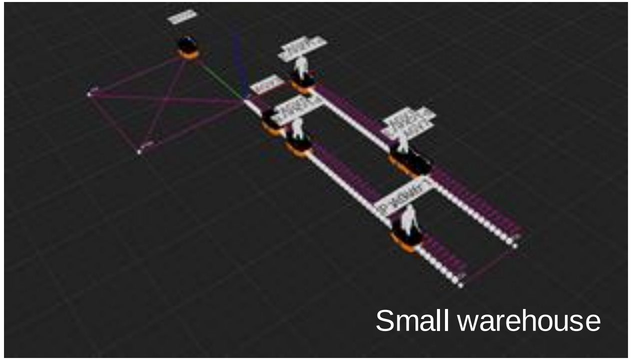
Scalable Multi-Agent Reinforcement Learning for Warehouse Logistics with Robotic and Human Co-Workers

DEMATIC

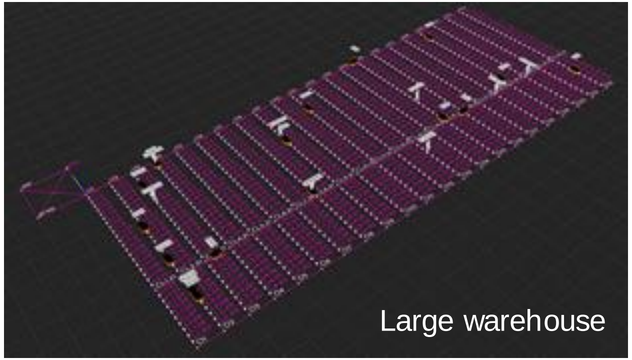
THE UNIVERSITY of EDINBURGH
informatics

Aleksandar Krnjaic, Raul D. Steleac,
Jonathan D. Thomas, Georgios Papoudakis,
Lukas Schäfer, Andrew Wing Keung To,
Kuan-Ho Lao, Murat Cubuktepe,
Matthew Haley, Peter Börsting,
Stefano V. Albrecht

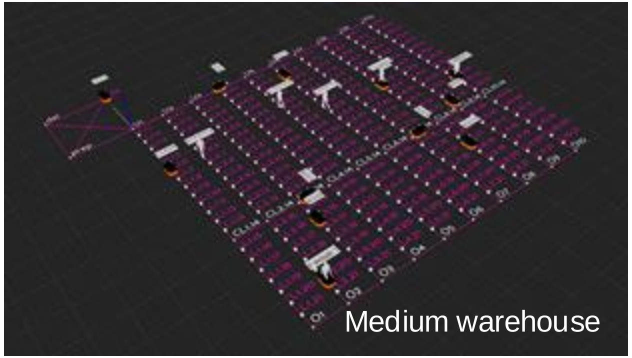
Warehouse Simulator



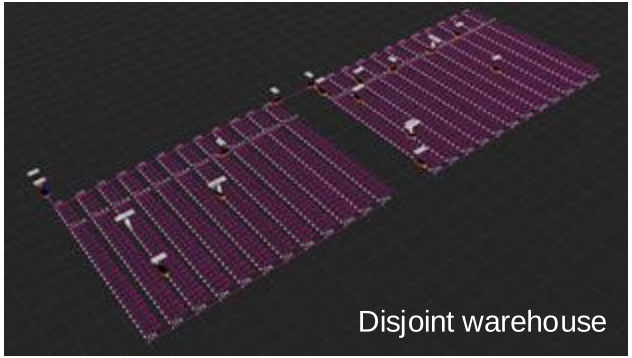
Small warehouse



Large warehouse



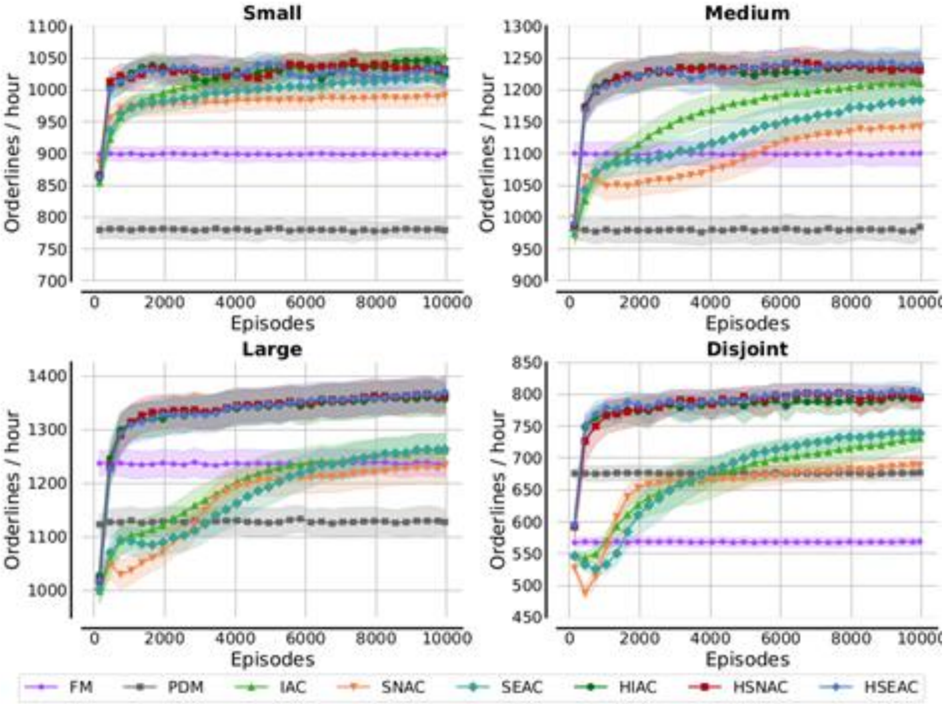
Medium warehouse



Disjoint warehouse

Evaluation Results

1. Comparison against the two industry leading heuristics:
    a. Follow Me (FM)
    b. Pick Don't Move (PDM)
2. Ablation study of the hierarchical module for three data sharing mechanisms:
    a. Independent Actor-Critic (IAC)
    b. Shared Network Actor-Critic (SNAC)
    c. Shared Experience Actor Critic (SEAC)



Part III: Algorithms in Practice

Implementing a MARL algorithm in PyTorch

Implementing MARL Algorithms

https://github.com/marl-book/codebase

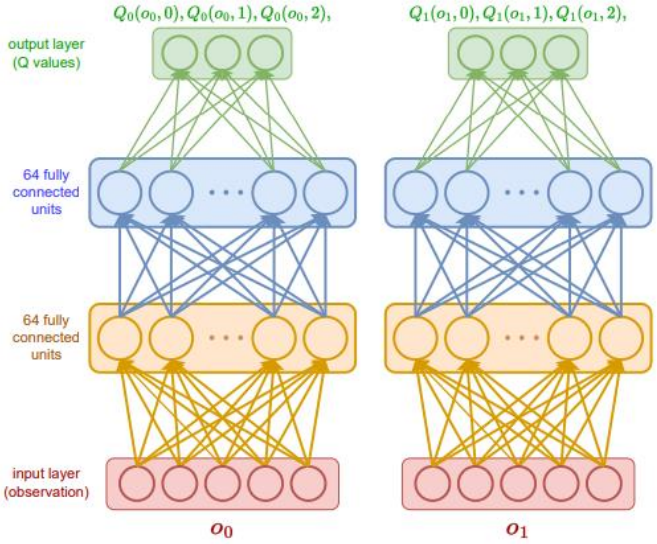
```
import lbforaging
import gym

env = gym.make("Foraging-8x8-2p-1f-v2")
```

```
env.observation_space
>> Tuple(Box(..., 15), Box(..., 15))

env.action_space
>> Tuple(Discrete(6), Discrete(6))
```

```
observations = env.reset()
next_observations, rewards, terminal_signal, _ = env.step(
    actions)
```

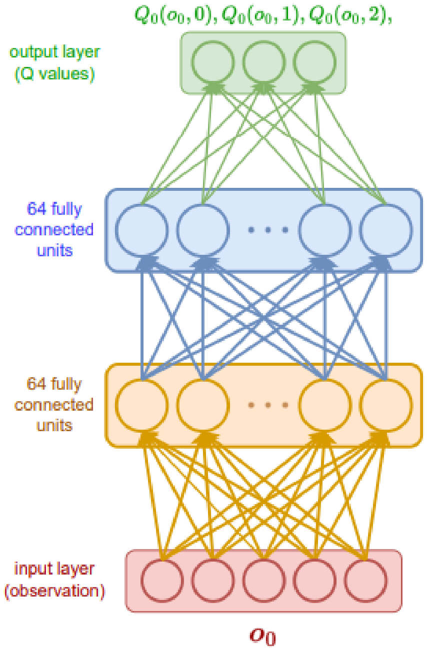
MARL: Neural Networks

```python
import torch
from torch import nn
from typing import List

class MultiAgentFCNetwork(nn.Module):
    def __init__(
            self,
            in_sizes: List[int],
            out_sizes: List[int]
        ):
        super().__init__()

        # We use the ReLU activation function:
        activ = nn.ReLU
        # We use two hidden layers of 64 units each:
        hidden_dims = (64, 64)

        n_agents = len(in_sizes)
        # The number of agents is the length of the
        # input and output vector
        assert n_agents == len(out_sizes)

        # We will create 'n_agents' (independent) networks
        self.networks = nn.ModuleList()

        # For each agent:
        for in_size, out_size in zip(in_sizes, out_sizes):
            network = [
                nn.Linear(in_size, hidden_dims[0]),
                activ(),
                nn.Linear(hidden_dims[0], hidden_dims[1]),
                activ(),
                nn.Linear(hidden_dims[1], out_size),
            ]
            self.networks.append(nn.Sequential(*network))

    def forward(self, inputs: List[torch.Tensor]):

        # The networks can run in parallel:
        futures = [
            torch.jit.fork(model, inputs[i])
                for i, model in enumerate(self.networks)
        ]
        results = [torch.jit.wait(fut) for fut in futures]
        return results
```



Seamless Parameter Sharing Implementation

```python
class MultiAgentFCNetwork_SharedParameters(nn.Module):

    def __init__(
            self,
            in_sizes: List[int],
            out_sizes: List[int]
        ):

        # ... same as MultiAgentFCNetwork

        # We will create one (shared) network
        # This assumes that input and output size of the
        # networks is identical across agents. If not, one
        # could first pad the inputs and outputs

        network = [
            # ... same as MultiAgentFCNetwork
        ]
        self.network = nn.Sequential(*network)

    def forward(self, inputs: List[torch.Tensor]):

        # A forward pass of the same network in parallel
        futures = [
            torch.jit.fork(self.network, inp)
                for inp in inputs)
        ]
        results = [torch.jit.wait(fut) for fut in futures]
        return results
```



Initialising and Querying the Models

```python
# Example of observation of agent 1:
# obs1 = torch.tensor([1, 0, 2, 3, 0])

# Example of observation of agent 2:
# obs2 = torch.tensor([0, 0, 0, 3, 0])

obs_sizes = (5, 5)

# Example of action of agent 1:
# act1 = [0, 0, 1] # one-hot encoded

# Example of action of agent 2:
# act2 = [1, 0, 0] # one-hot encoded

action_sizes = (3, 3)

model = MultiAgentFCNetwork(obs_sizes, action_sizes)

# Alternatively, the shared parameter model can be used instead:
# model = MultiAgentFCNetwork_SharedParameters(
#     obs_sizes, action_sizes
#)
```

```python
# obs1, obs2, model as above

q_values = model([obs1, obs2])
>> ([Q11, Q12, Q13], [Q21, Q22, Q23])
# where Qij is the Q value of agent i doing action j
```

Independent DQN

```
# obs1, obs2, model as above

q_values = model([obs1, obs2])
>> ([Q11, Q12, Q13], [Q21, Q22, Q23])
# where Qij is the Q value of agent i doing action j
```

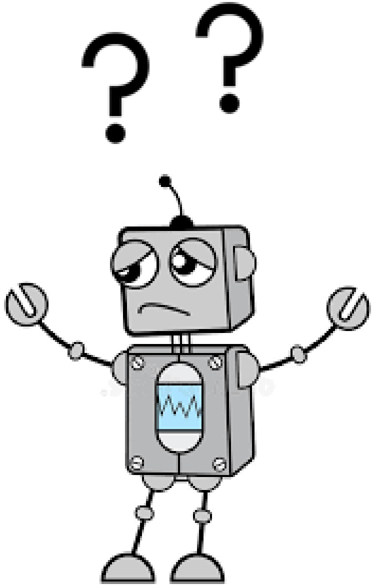
```
# we are creating a new "agent" dimension
q_values_stacked = torch.stack(q_values)
print(q_values_stacked.shape)
>> [2, 3]
# 2: agent dimension, 3: action dimension

# calculating best actions per agent (index):
_, a_prime = q_values_stacked.max(-1)
```

Optimising…

```
params_list = list(nn_agent1.parameters())
              + list(nn_agent2.parameters())
              + list(nn_agent3.parameters())
              + ...
common_optimiser = torch.optim.Adam(params_list)
...
loss = loss1 + loss2 + loss3 + ...
loss.backward()
common_optimizer.step()
```

Part IV: AI Agents

Task: "Create a model that predicts the optimal price for real estate properties based on features like location, size, and market trends.



We could have an LLM-based AI Agent

1. Make use of general LLMs as priors – they encode general enough priors
2. Allow agents to fix their internal structures for specific tasks

LLM-based Agent

Write Python code to... [task]

LLM-based Agent

Write Python code to... [task]

Zero-Shot

LLM-based Agent

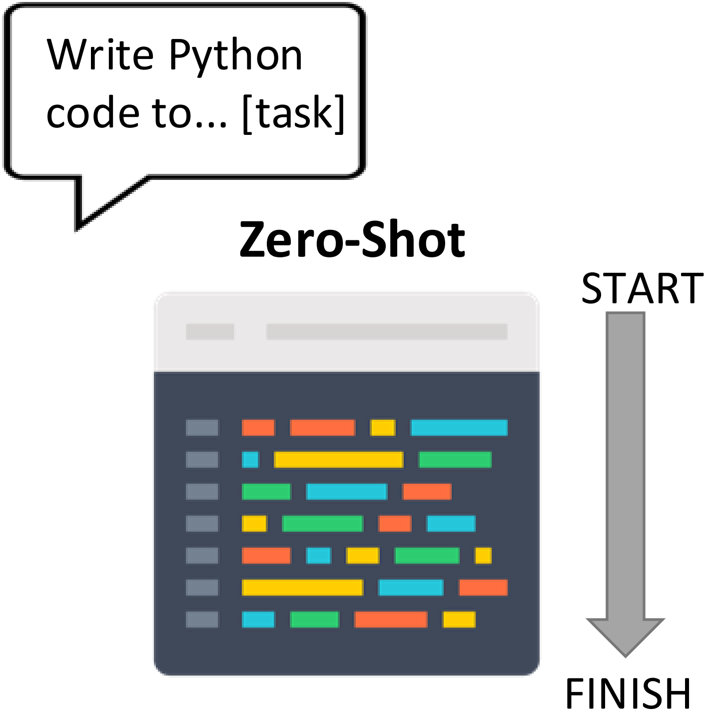
Write Python code to... [task]

Zero-Shot

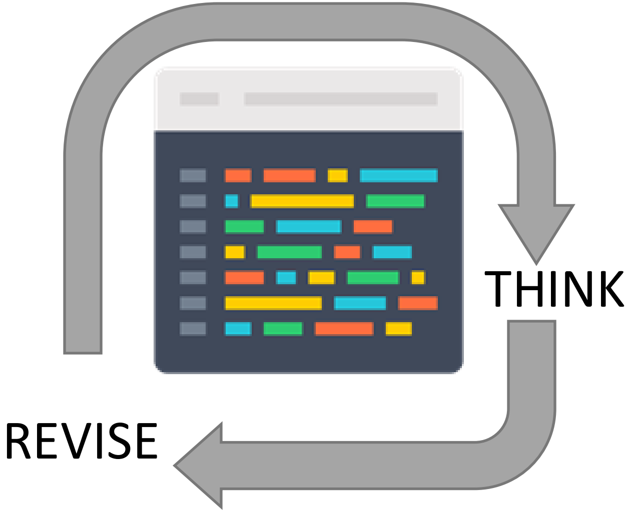
START

FINISH

LLM-based Agent



Agentic Workflow

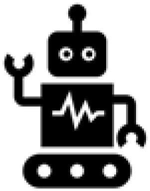


What are structuring techniques that people have been using?

Emerging components of an AI Agent

1. Planning
2. Reflection
3. Tools
4. Memory
5. Multi-Agent

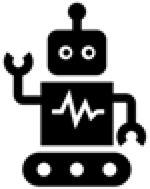
Planning



Recommended reading:
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.
Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

Planning

Do a linear regression in NumPy in this dataset.

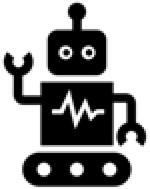


Recommended reading:
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.
Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

Planning

Do a linear regression in NumPy in this dataset.

And before starting, make a plan with bullet points…



Recommended reading:

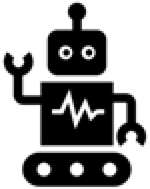
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.

Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

Planning

Do a linear regression in NumPy in this dataset.

And before starting, make a plan with bullet points…



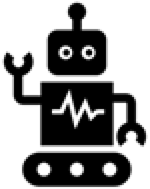
Agent makes a plan:

Recommended reading:
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.
Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

Planning

Do a linear regression in NumPy in this dataset.

And before starting, make a plan with bullet points…



Agent makes a plan:

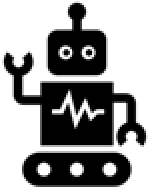
1. Download and process the dataset
2. Find and understand useful features
3. Write a draft of the code and run it with Python
4. ….

Recommended reading:
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.
Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

Planning

Do a linear regression in NumPy in this dataset.

And before starting, make a plan with bullet points…



Agent makes a plan:

1. Download and process the dataset
2. Find and understand useful features
3. Write a draft of the code and run it with Python
4. ….

And then the agent executes the plan

Recommended reading:
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.
Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

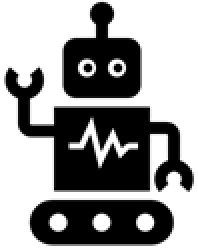
Planning: Why does it work?

1. Download and process the dataset
2. Find and understand useful features
3. Write a draft of the code and run it with Python
4. ….

Recommended reading:
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Wei et al.
Self-Consistency Improves Chain of Thought Reasoning in Language Models. Wang et al.

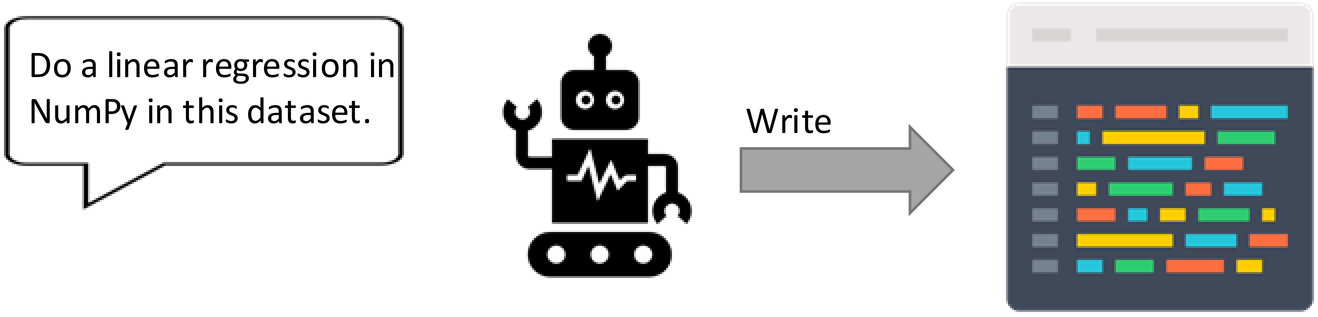
Reflection

Do a linear regression in NumPy in this dataset.



Recommended reading:
Reflexion: Language Agents with Verbal Reinforcement Learning. Shinn et al.

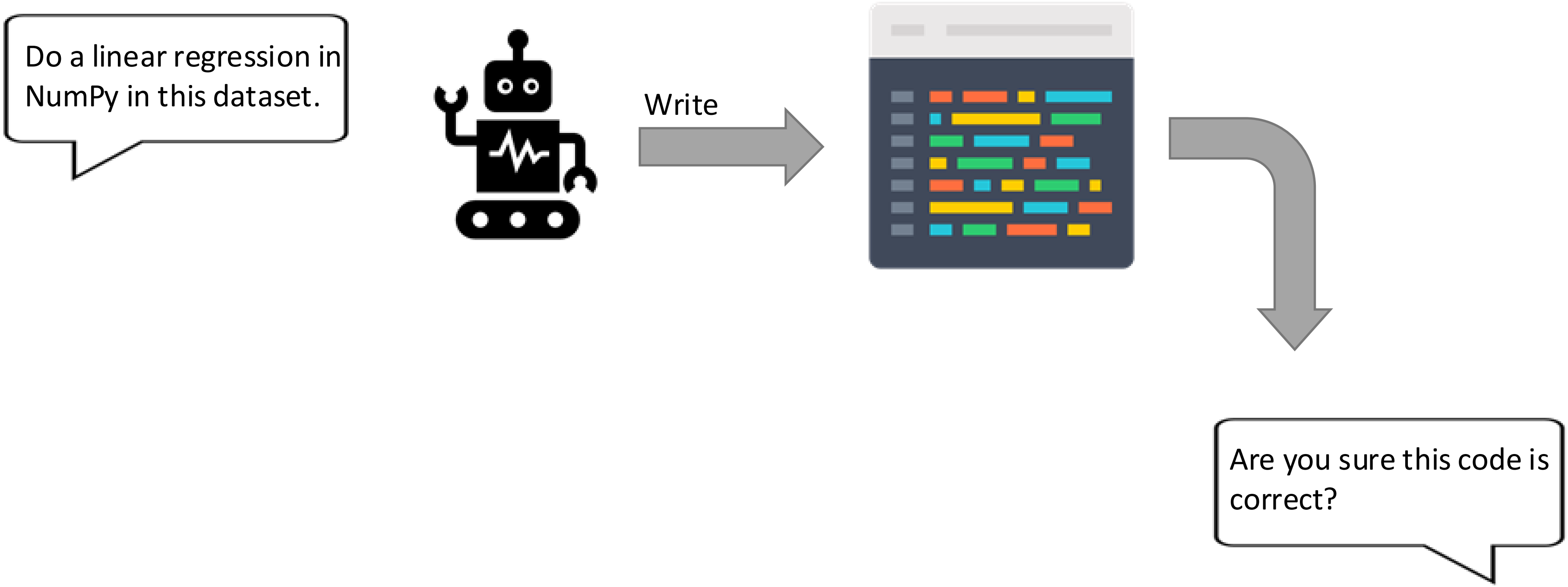
Reflection



Recommended reading:

Reflexion: Language Agents with Verbal Reinforcement Learning. Shinn et al.

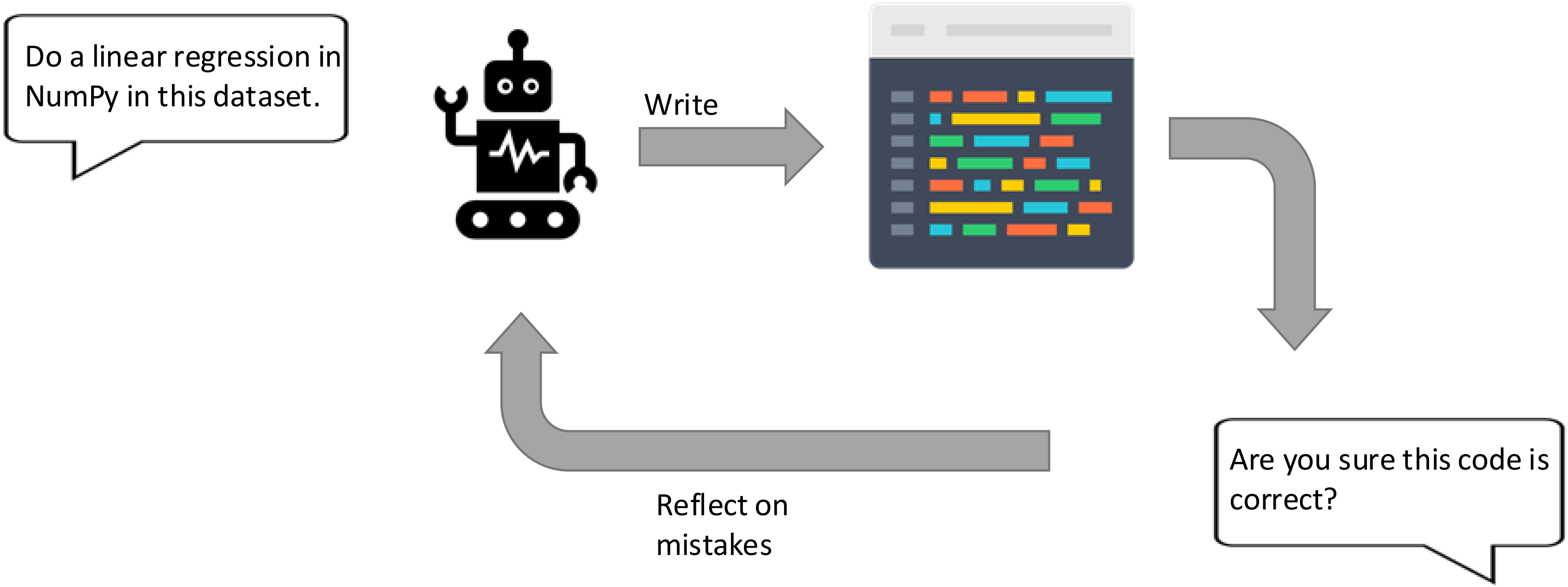
Reflection



Recommended reading:

Reflexion: Language Agents with Verbal Reinforcement Learning. Shinn et al.

Reflection

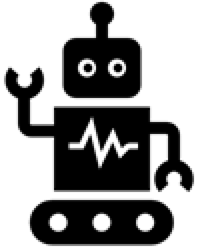


Recommended reading:

Reflexion: Language Agents with Verbal Reinforcement Learning. Shinn et al.

Memory

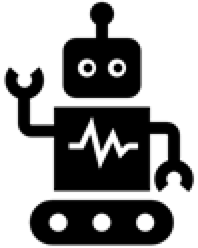
Do a linear regression in NumPy in this dataset.



Recommended reading:
Enhancing Large Language Models with Long-Term Memory. Zhong et al.

Memory

Do a linear regression in NumPy in this dataset.

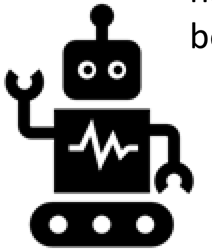


Previous AI code, examples, human code

Recommended reading:
Enhancing Large Language Models with Long-Term Memory. Zhong et al.

Memory

Do a linear regression in NumPy in this dataset.

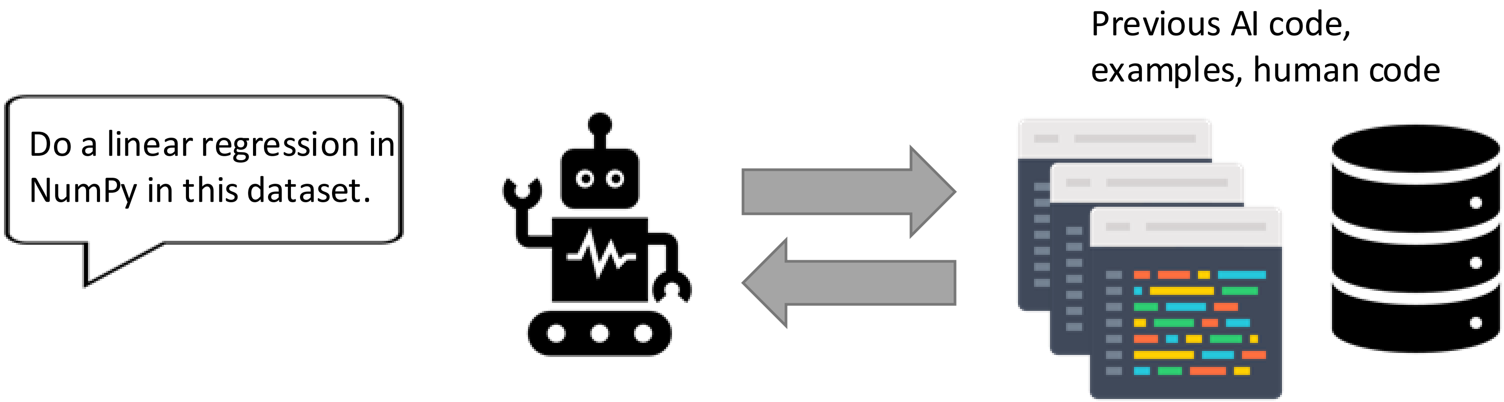


Has anything similar happened before?

Previous AI code, examples, human code

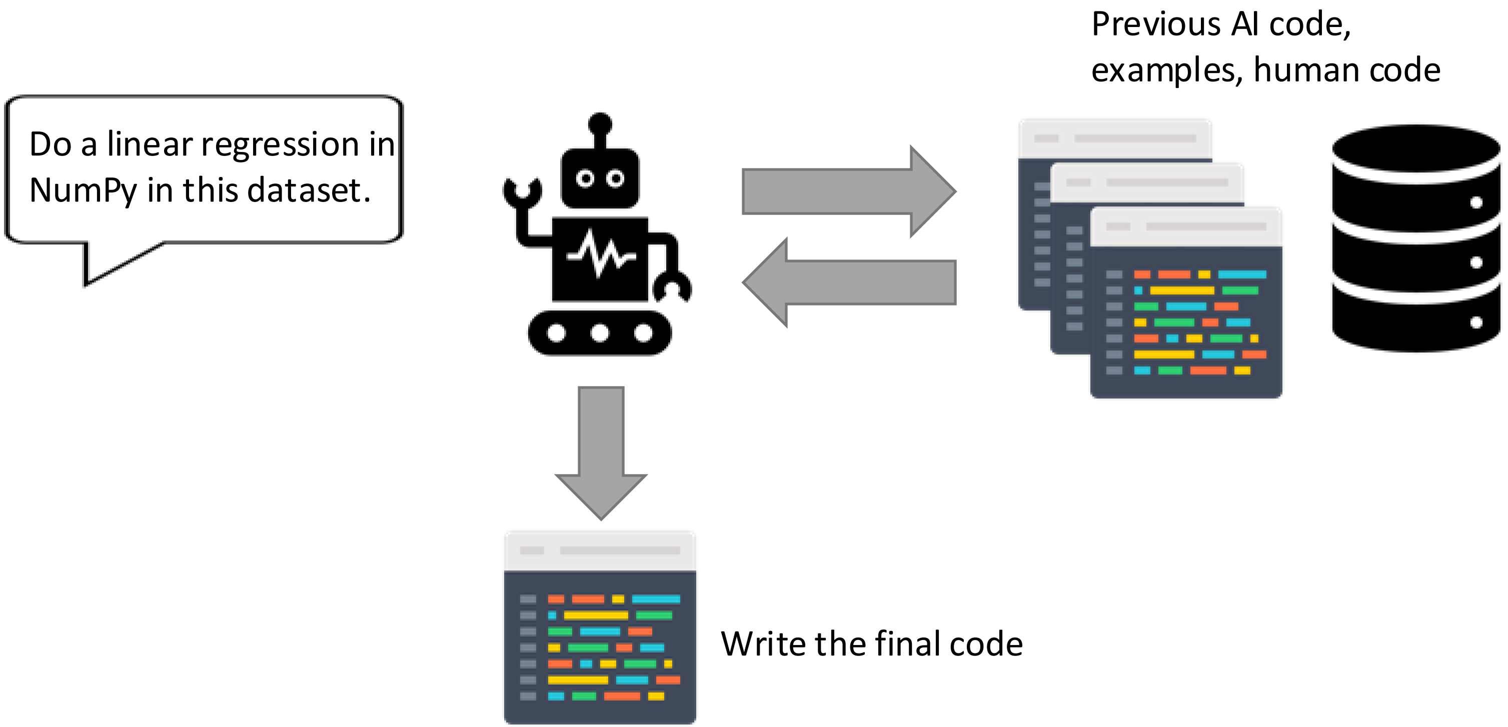
Recommended reading:
Enhancing Large Language Models with Long-Term Memory. Zhong et al.

Memory



Recommended reading:
Enhancing Large Language Models with Long-Term Memory. Zhong et al.

Memory



Recommended reading:
Enhancing Large Language Models with Long-Term Memory. Zhong et al.

Multi-Agent

Do a linear regression in NumPy in this dataset.

Coder Agent

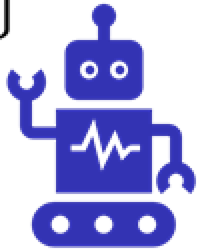
Critic Agent

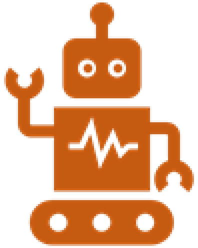
Recommended reading:
ChatDev: Communicative Agents for Software Development. Qian et al.

Multi-Agent

Do a linear regression in NumPy in this dataset.

Coder Writes the code:





Coder Agent

Critic Agent

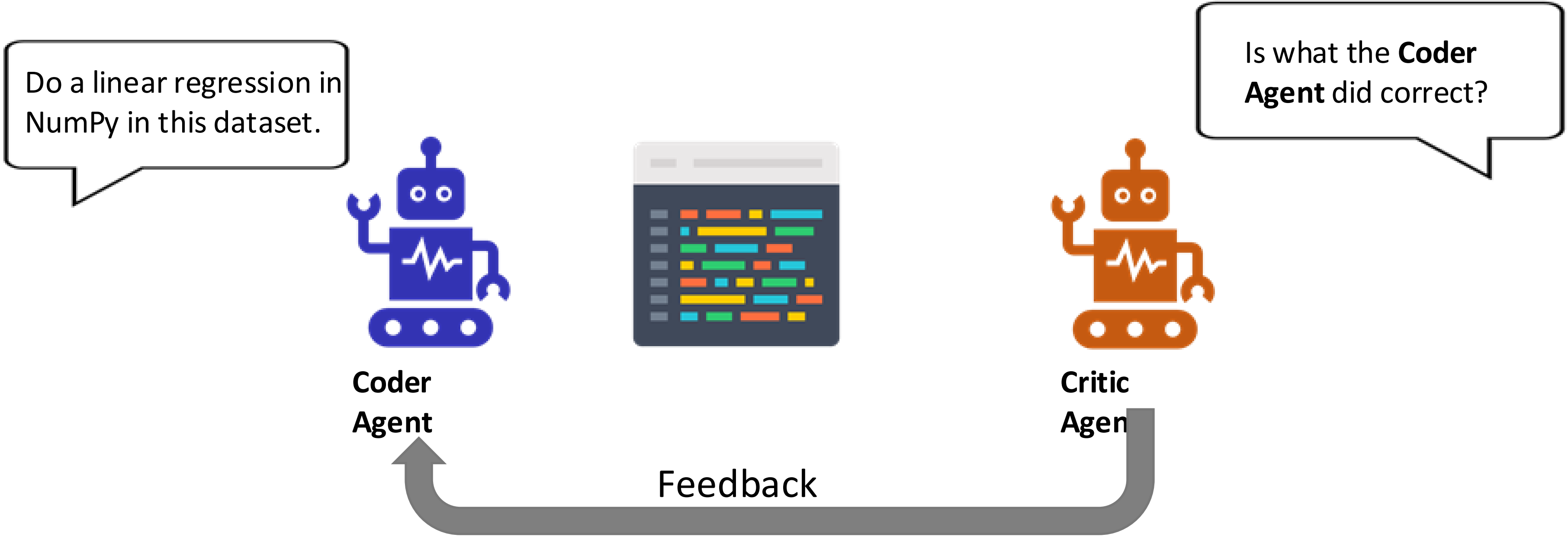
Recommended reading:

ChatDev: Communicative Agents for Software Development. Qian et al.

Multi-Agent

Recommended reading:
ChatDev: Communicative Agents for Software Development. Qian et al.

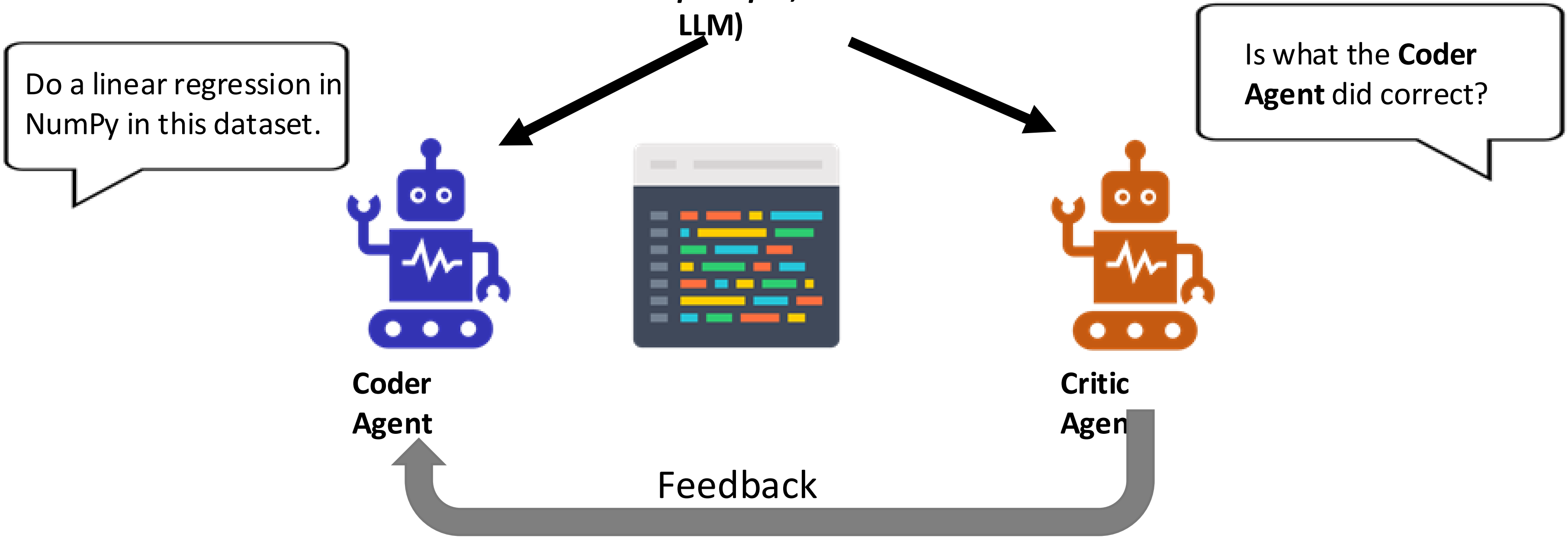
Multi-Agent



Recommended reading:
ChatDev: Communicative Agents for Software Development. Qian et al.

Multi-Agent

Each Agent might have different components! (*memories*, *tools*, *prompts*, even different LLM)

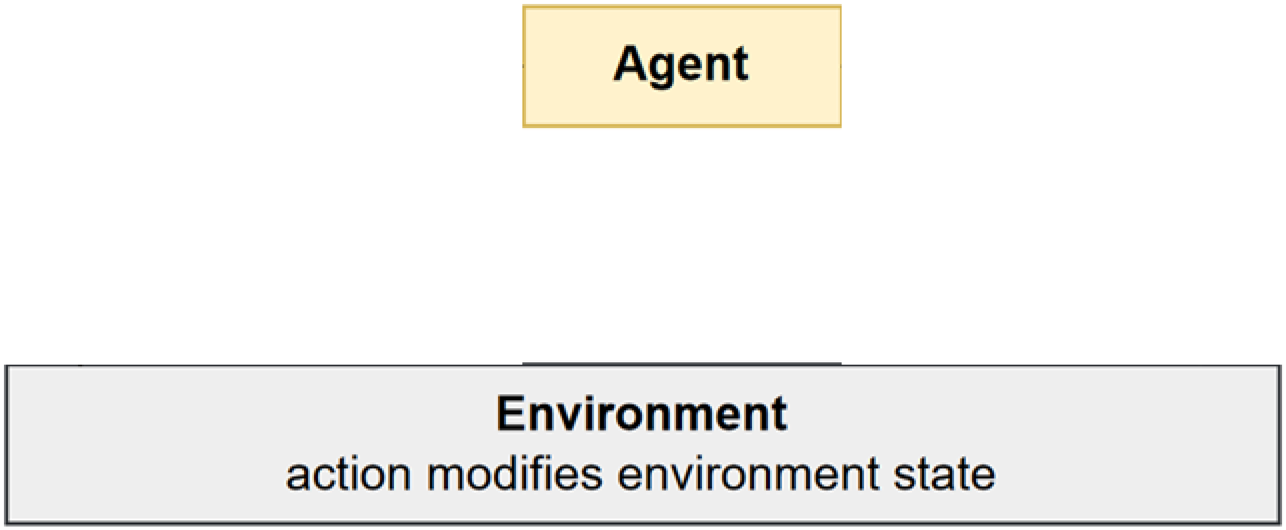


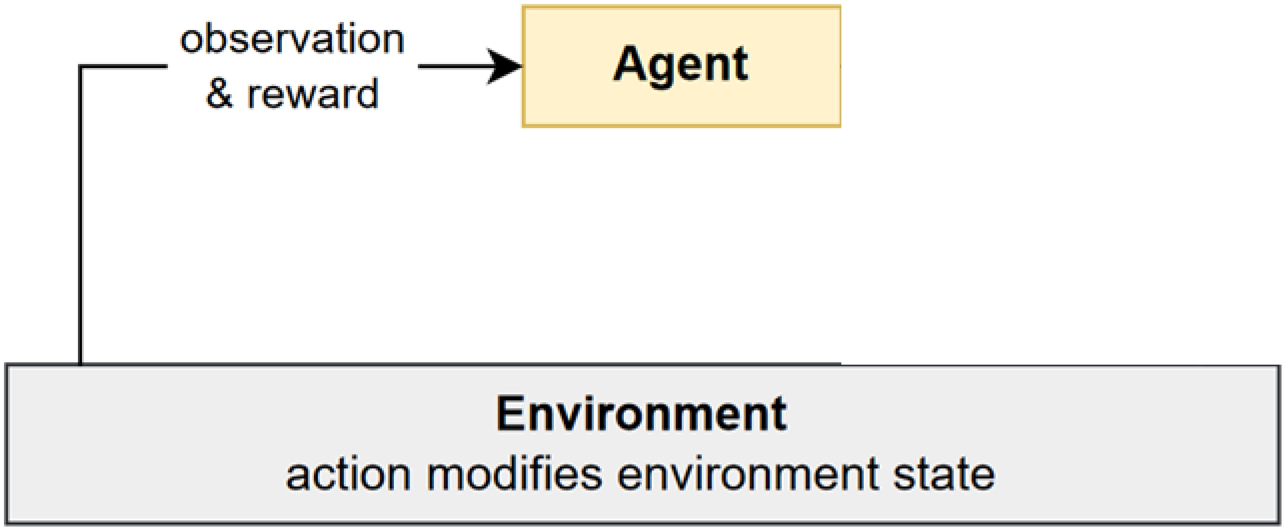
Recommended reading:
ChatDev: Communicative Agents for Software Development. Qian et al.

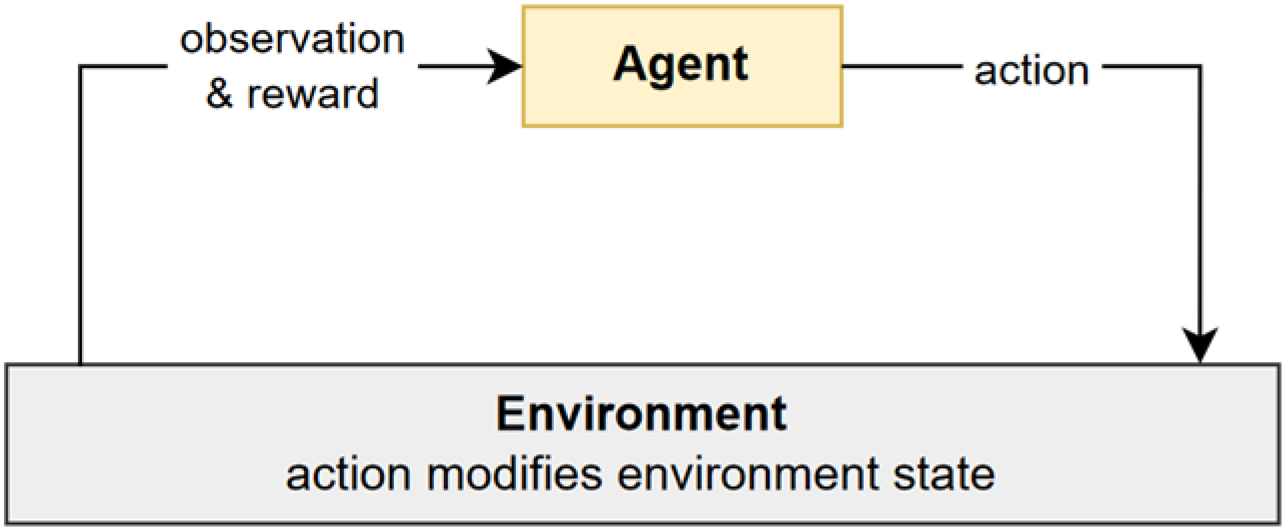
How do we put all these together to create powerful AI Agents?

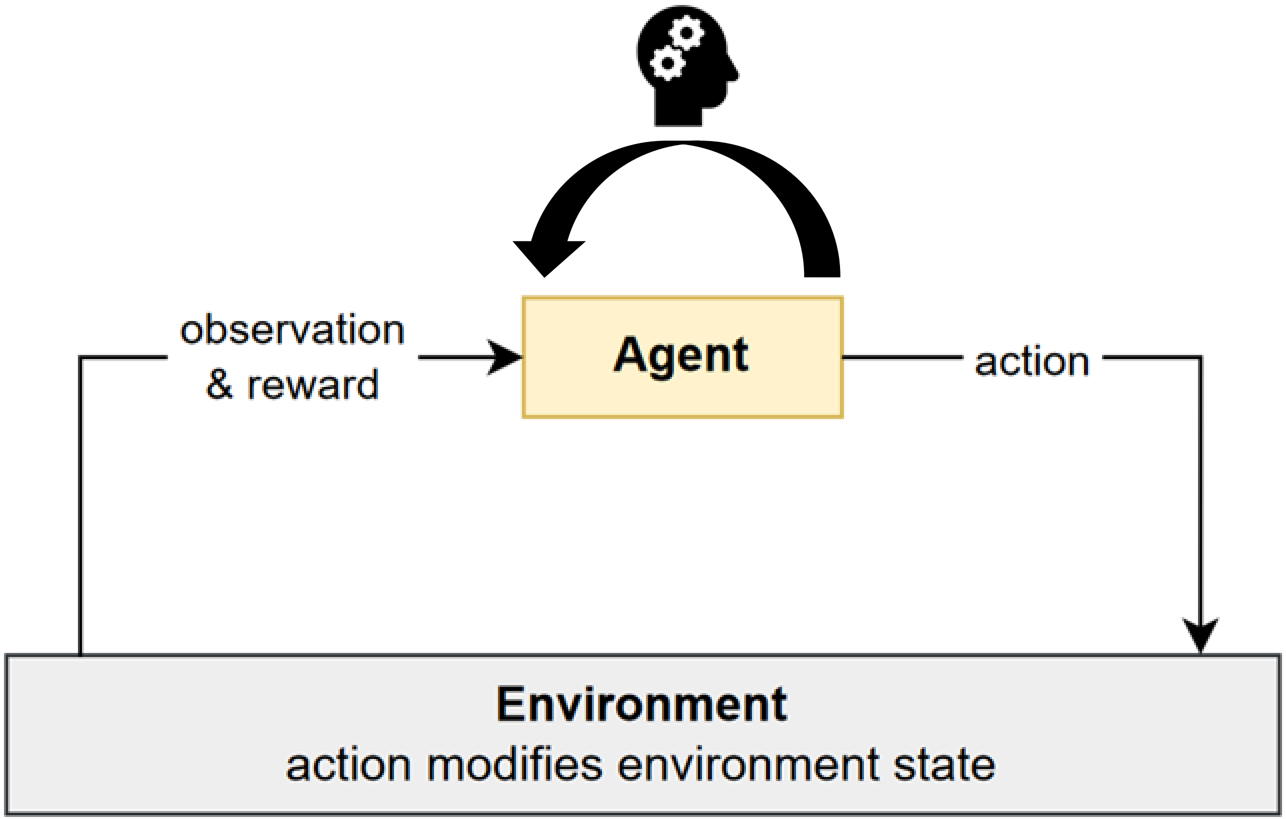
Pangu-Agent: A Fine-Tunable Generalist Agent with Structured Reasoning

Filippos Christianos, Georgios Papoudakis, Matthieu Zimmer, Thomas Coste, Zhihao Wu, Jingxuan Chen, Khyati Khandelwal, James Doran, Xidong Feng, Jiacheng Liu, Zheng Xiong, Yicheng Luo, Jianye Hao, Kun Shao, Haitham Bou-Ammar, Jun Wang


Agent
Environment
action modifies environment state


observation
& reward
Agent
Environment
action modifies environment state


observation
& reward
Agent
action
Environment
action modifies environment state


observation
& reward
Agent
action
Environment
action modifies environment state

What is our objective?

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[G|\pi]$$

$$\underset{\pi}{\operatorname{argmax}} \, \mathbb{E}[G|\pi]$$

1. Solutions are too task specific requiring lots of engineering

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[G|\pi]$$

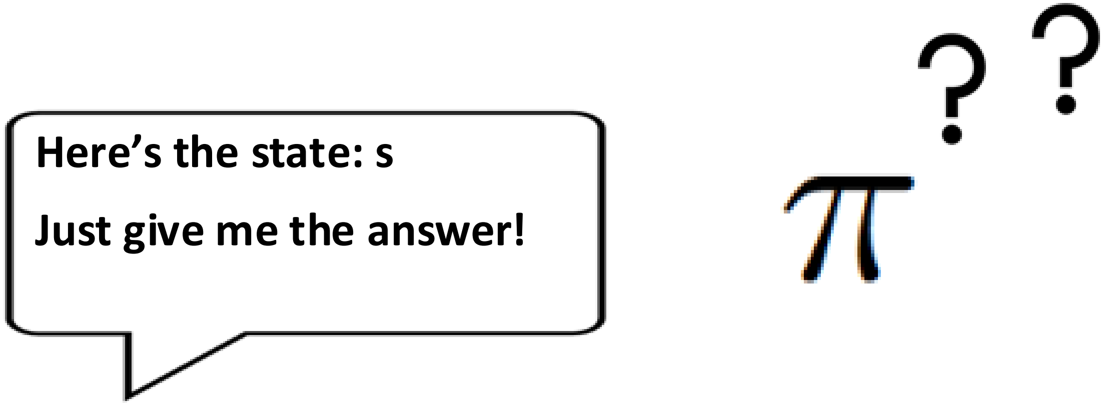
1. Solutions are too task specific requiring lots of engineering

2. Agent structures are pre-defined and can’t adapt their reasoning to suit tasks

Here’s the state: s

Just give me the answer!

π


Here’s the state: s
Just give me the answer!

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[G|\pi]$$

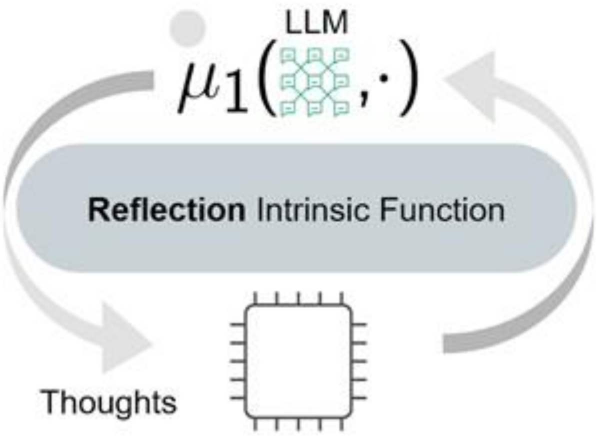
$$\operatorname*{argmax}_{\pi} \mathbb{E}[G|\pi] \Rightarrow \operatorname*{argmax}_{\pi,\mu} \mathbb{E}[G|\pi, \mu]$$

$$\underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[G|\pi] \Rightarrow \underset{\pi,\mu}{\operatorname{argmax}}\, \mathbb{E}[G|\pi, \mu]$$

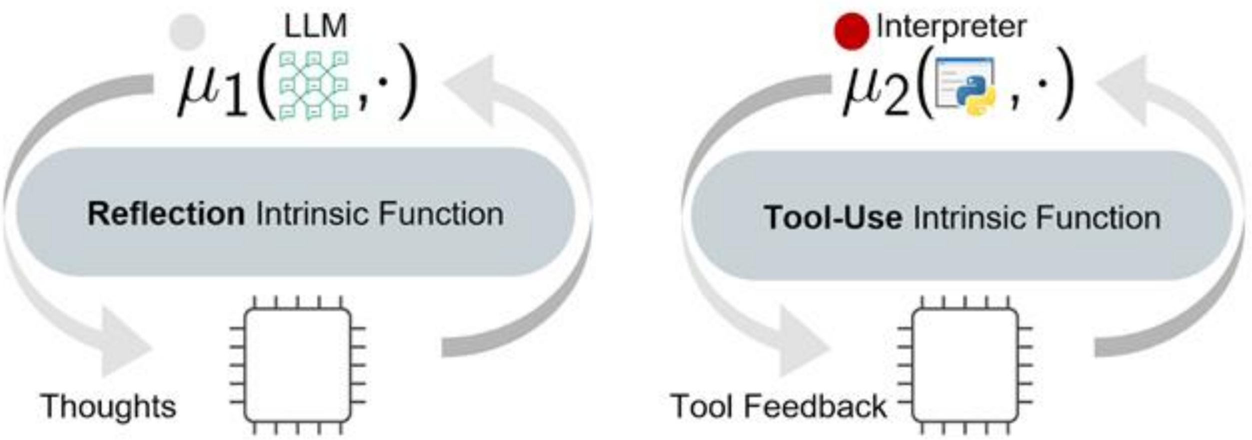
Structured reasoning: intrinsic functions!

- *Intrinsic function (μ)*: one that operates on memory (short term or long term)

- *Intrinsic function*: one that operates on memory (short term or long term)

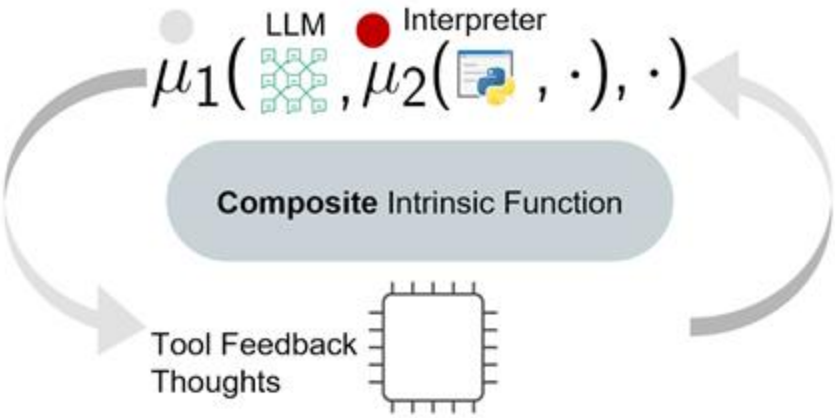


- *Intrinsic function*: one that operates on memory (short term or long term)

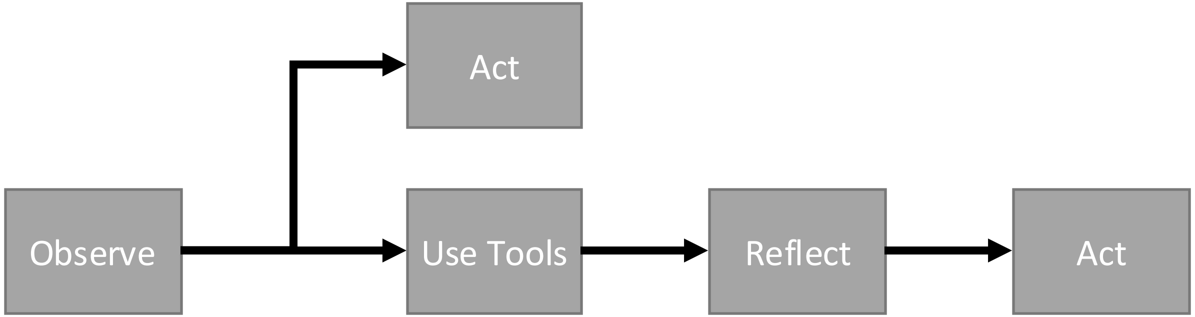


- *Intrinsic function*: one that operates on memory (short term or long term)
- **We can make *composite functions* by *nesting* intrinsic functions**

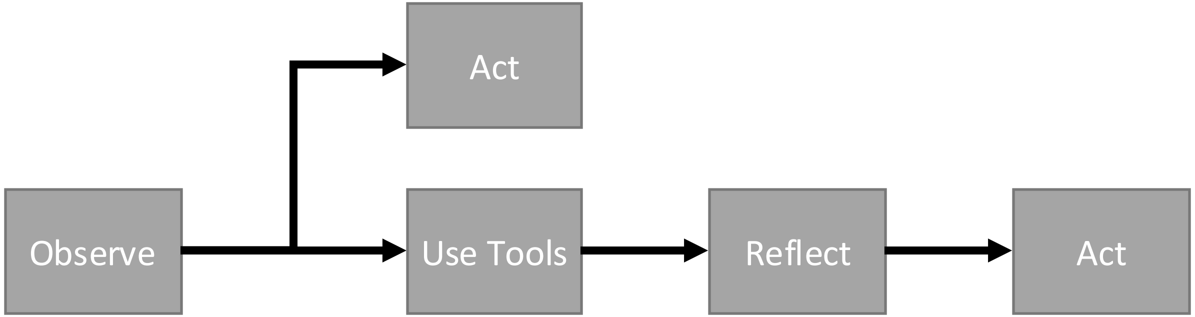
- *Intrinsic function*: one that operates on memory (short term or long term)
- We can make *composite functions* by *nesting* intrinsic functions



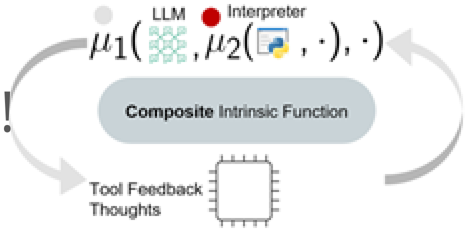
Which means we can make agentic workflows…



Which means we can make agentic workflows…



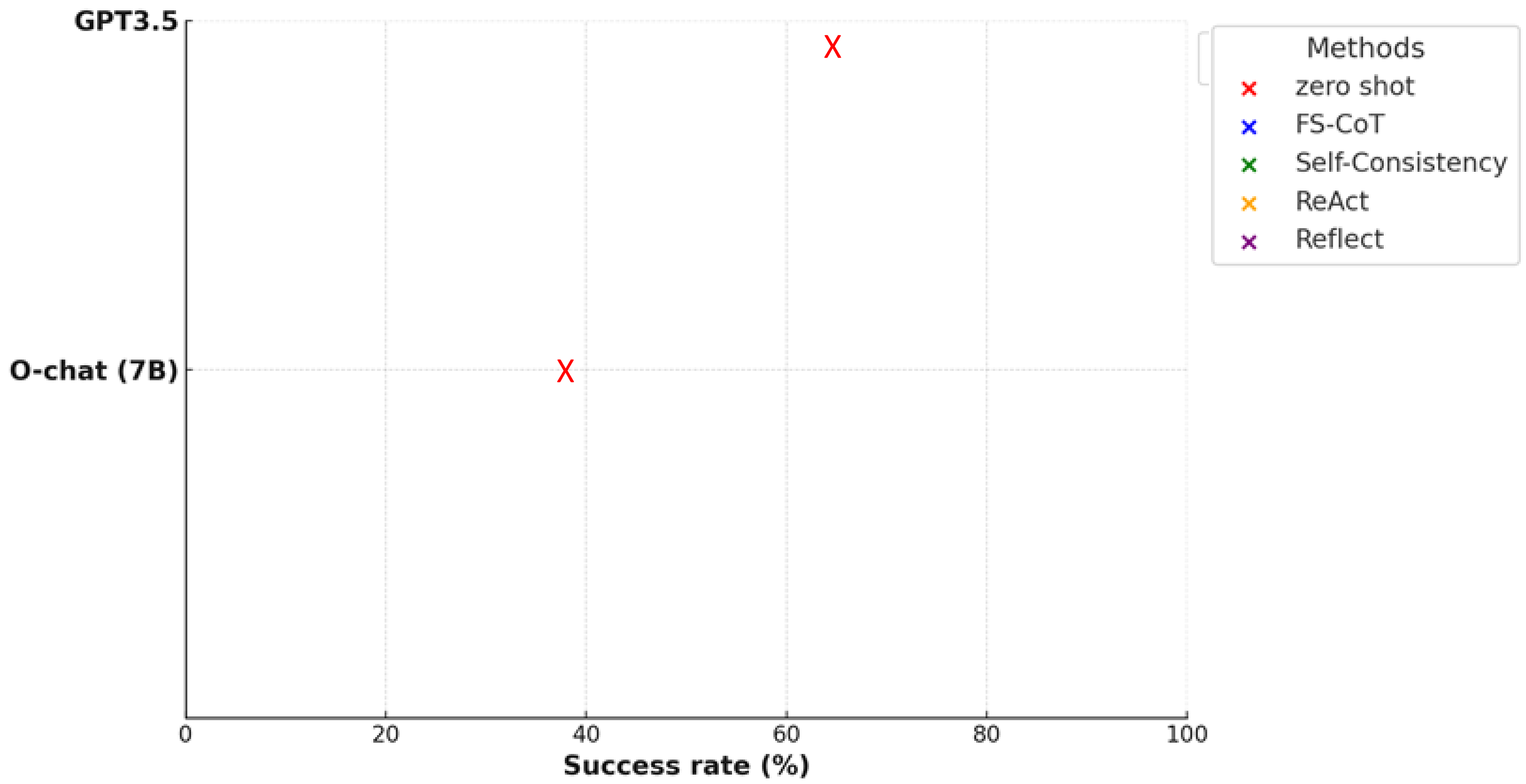
Still just a composite function!



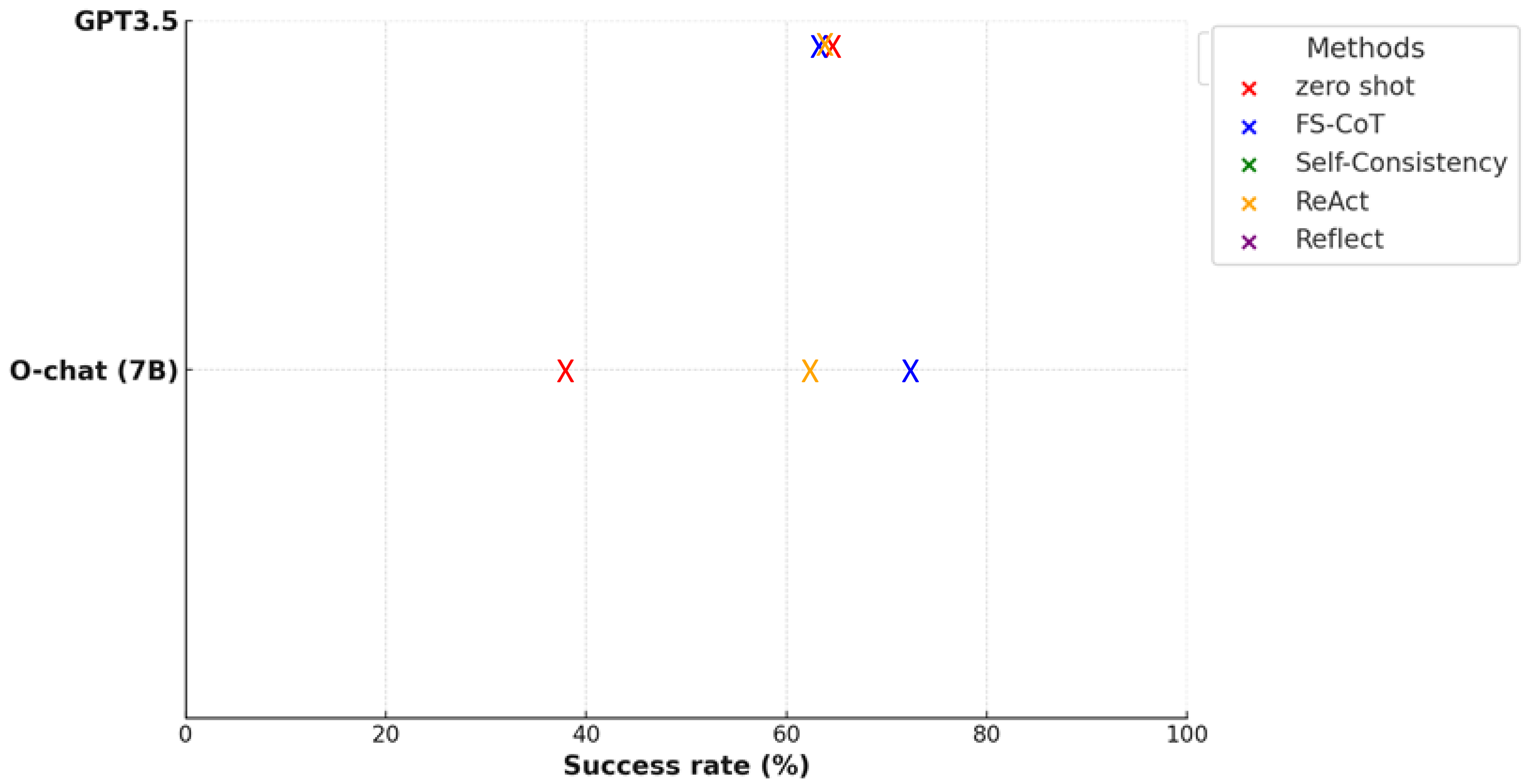
… And this encompasses everything we have talked about so far…

… And this encompasses everything we have talked about so far…

Pangu-Agent is built with this formulation in mind

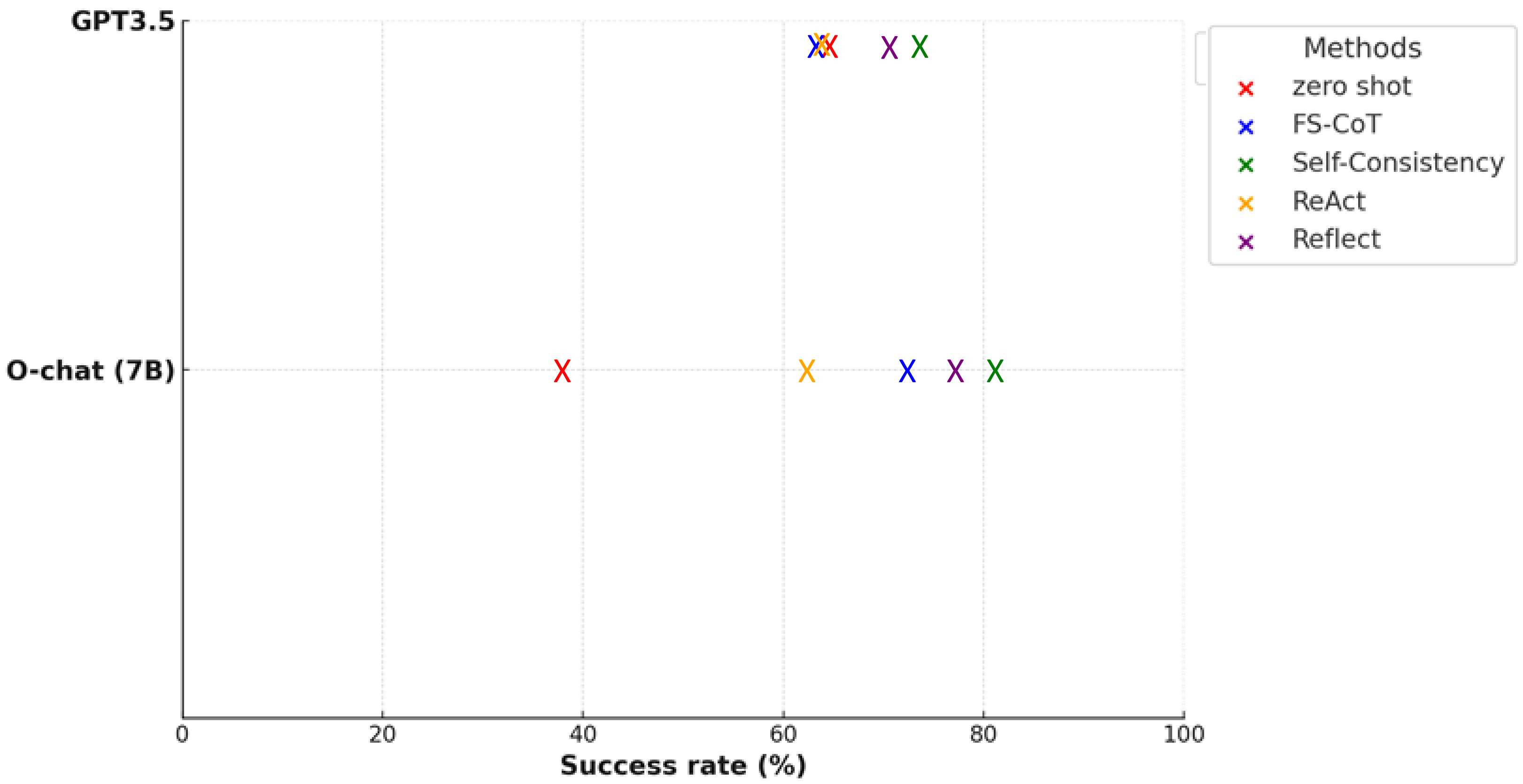


Results on GSM8K



Results on GSM8K

And we can implement more complex methods (composite) function too..



Results on GSM8K

The Pangu Agent framework:

The Pangu Agent framework:

1. Can use various LLMs. LLMs is just an input to the intrinsic function…

The Pangu Agent framework:

1. Can use various LLMs. LLMs is just an input to the intrinsic function…
2. It can be used in different environments/tasks.

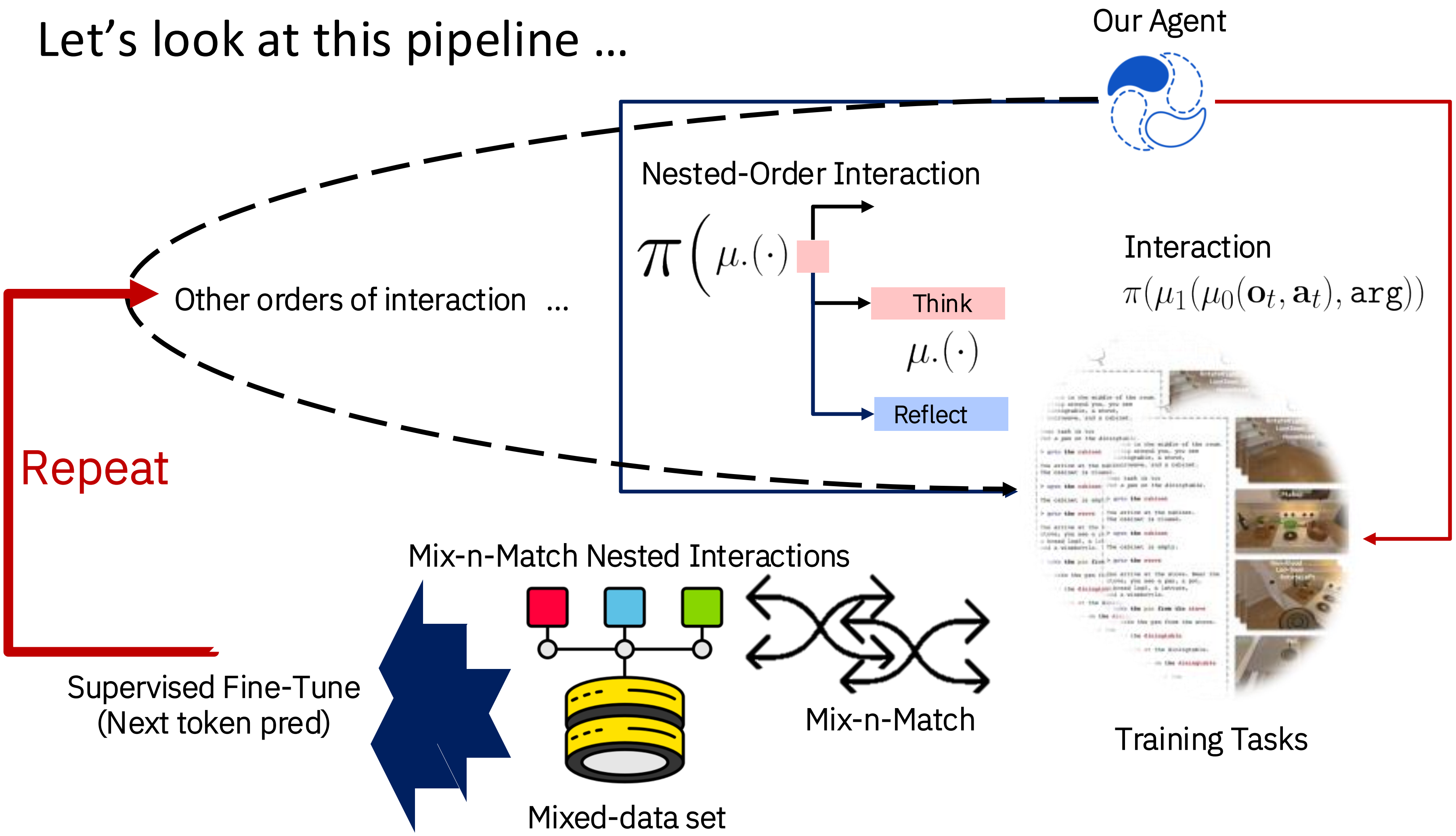
The Pangu Agent framework:

1. Can use various LLMs. LLMs is just an input to the intrinsic function…
2. It can be used in different environments/tasks.
3. It can be used with a variety of methods… (including existing ones)

But that’s not all… Remember $\underset{\pi,\mu}{\operatorname{argmax}}\, \mathbb{E}[G|\pi,\mu]$

ALFWorld

Let’s look at this pipeline …



Tasks	Openchat v3.5				
	Direct	**FS-CoT**	**1-step SFT**	**2-step SFT**	**3-step SFT**
ALFWorld	0.04	0.22	0.45	0.68	**0.82**

Table 4: Benchmark of Openchat v3.5 with/without fine-tuning on held-out tasks.

And then RL…

Tasks	Opencht v3.5		Llama-2-7B			
	Direct	FS-CoT	Original	SFT	SFT+RL	RL
ALFWorld	0.04	0.22	0	0.5	**0.88**	0.04
BabyAI-GoToObj-v0	0.31	0.61	0.28	0.75	**0.91**	0.87
BabyAI-GoToRedBlueBall-v0	0.11	0.43	0.04	0.21	**0.77**	0.69

Table 5: Benchmark of Openchat and LLama-2-7b with/without fine-tuning on held-out tasks.


Openchat SFT
Task: Put clean mug
in coffeemachine
Press q to Exit

Part V: App Control with AI Agents

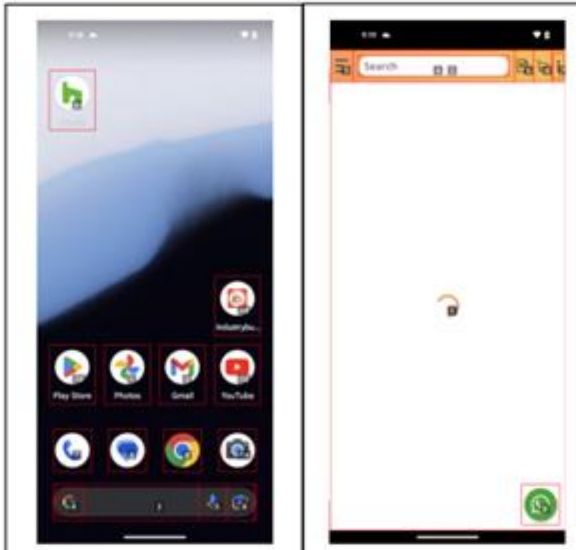
GOAL: I’d like to get a new three-seater sofa for Christmas because my old one broke – search in the “industry buy” app.

GOAL: I’d like to get a new three-seater sofa for Christmas because my old one broke – search in the “industry buy” app.

GOAL: I’d like to get a new three-seater sofa for Christmas because my old one broke – search in the “industry buy” app.

Correct action:
{'action_type': 'open_app', 'app_name': 'Industrybuying'}

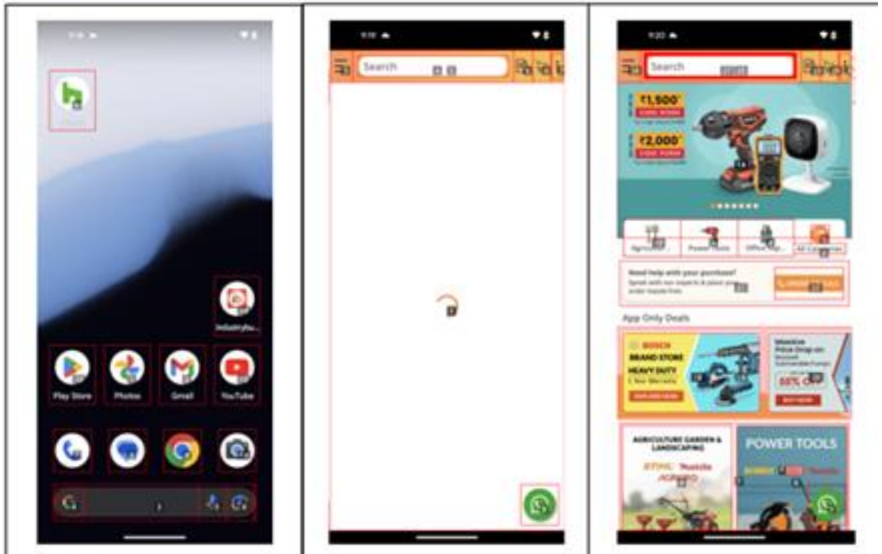
GOAL: I’d like to get a new three-seater sofa for Christmas because my old one broke – search in the “industry buy” app.



Correct action:
{'action_type': 'open_app', 'app_name': 'Industrybuying'}

Correct action:
{'action_type': 'wait'}

GOAL: I’d like to get a new three-seater sofa for Christmas because my old one broke – search in the “industry buy” app.

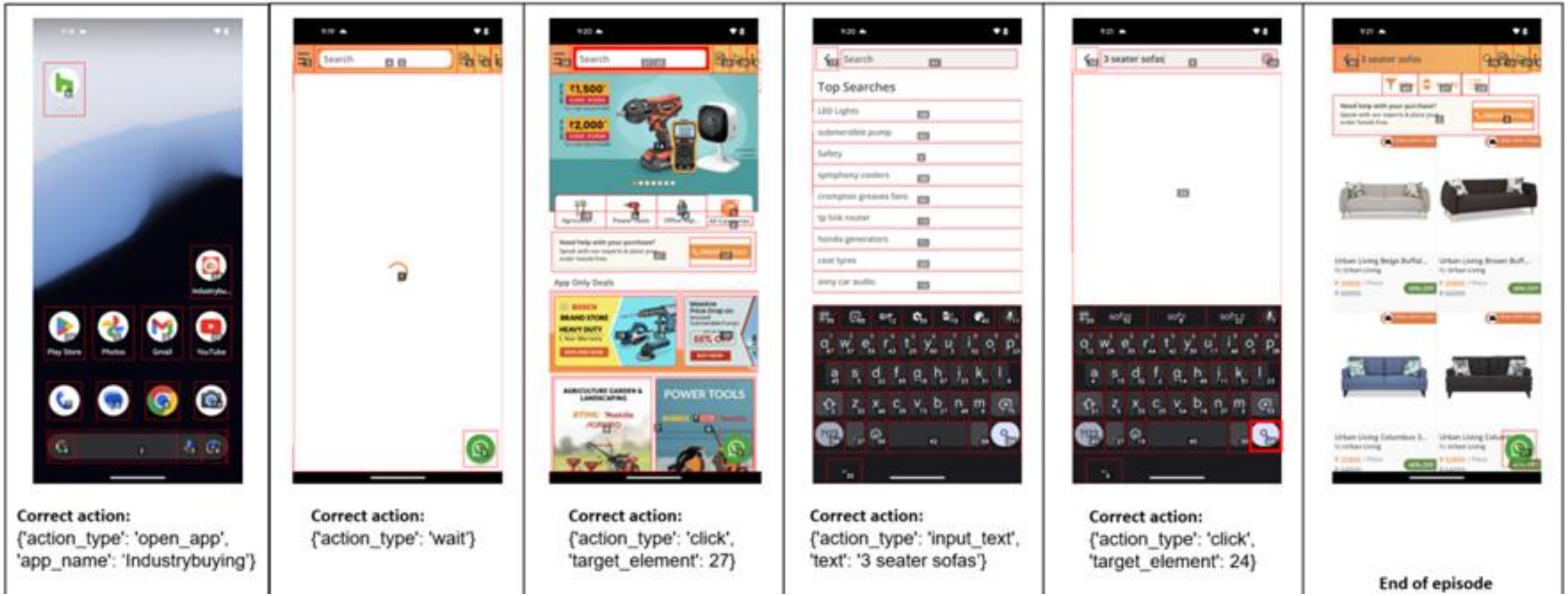


Correct action:
{'action_type': 'open_app', 'app_name': 'Industrybuying'}

Correct action:
{'action_type': 'wait'}

Correct action:
{'action_type': 'click', 'target_element': 27}

GOAL: I’d like to get a new three-seater sofa for Christmas because my old one broke – search in the “industry buy” app.



Training Datasets

1. Android in the Wild (AitW): 17k* episodes
2. Android Control: 15k episodes

Training Datasets

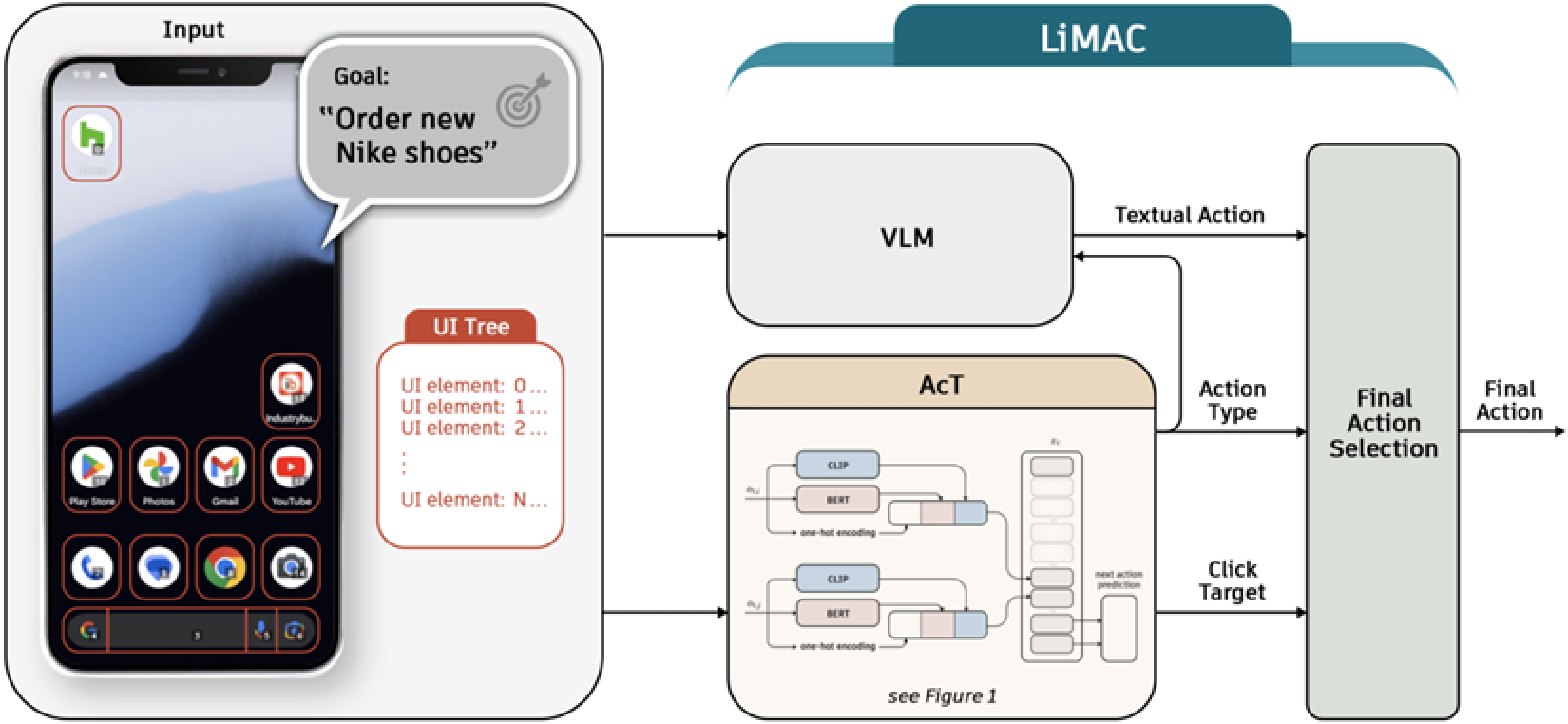
1. Android in the Wild (AitW): 17k* episodes
2. Android Control: 15k episodes

Each episode contains:

- Goal
- Observations (screenshot, or UI tree)
- Actions (click, type, wait, etc…)

Motivation

1. Low Computational requirements
2. Efficient preprocessing of input screenshot
3. Condition action in past trajectory
4. Specialisation towards click actions


Input
Goal:
"Order new Nike shoes"
UI Tree
UI element: 0 ...
UI element: 1 ...
UI element: 2 ...
UI element: N ...
LiMAC
VLM
Textual Action
AcT
see Figure 1
Action Type
Click Target
Final Action Selection
Final Action

Approach

1. Crop each UI element and embed it into one token
   - each observation becomes a list of tokens

Approach

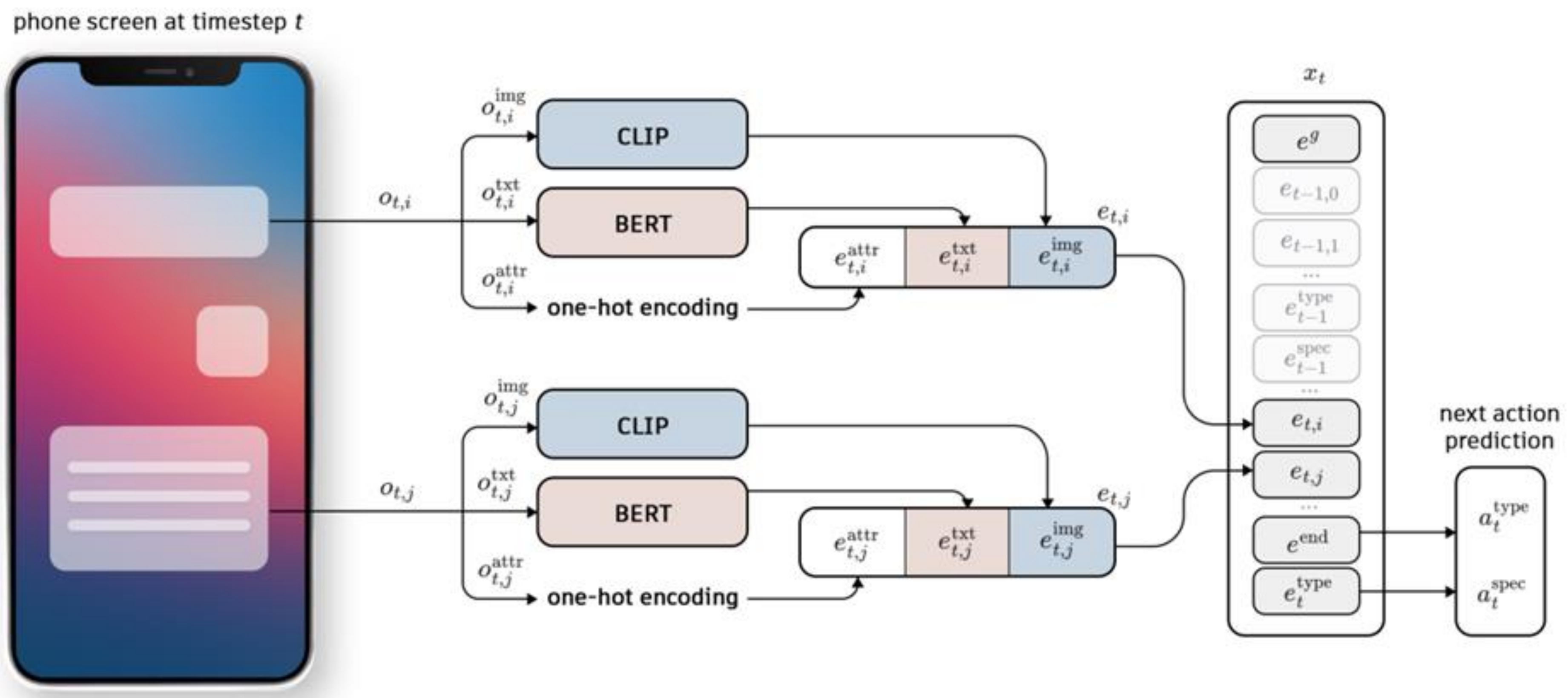
1. Crop each UI element and embed it into one token
    - each observation becomes a list of tokens
2. Formulate the action type selection as a classification problem
    - train a small, fast, transformer called AcT

Approach

1. Crop each UI element and embed it into one token
   - each observation becomes a list of tokens
2. Formulate the action type selection as a classification problem
   - train a small, fast, transformer called AcT
3. Use the same model with contrastive learning to predict the target of click actions

Approach

1. Crop each UI element and embed it into one token
    - each observation becomes a list of tokens
2. Formulate the action type selection as a classification problem
    - train a small, fast, transformer called AcT
3. Use the same model with contrastive learning to predict the target of click actions
4. Use a small VLM for actions that require text



phone screen at timestep t
$o_{t,i}$
$o_{t,i}^{\mathrm{img}}$
CLIP
$o_{t,i}^{\mathrm{txt}}$
BERT
$o_{t,i}^{\mathrm{attr}}$
one-hot encoding
$e_{t,i}$
$e_{t,i}^{\mathrm{attr}}$ $e_{t,i}^{\mathrm{txt}}$ $e_{t,i}^{\mathrm{img}}$
$o_{t,j}$
$o_{t,j}^{\mathrm{img}}$
CLIP
$o_{t,j}^{\mathrm{txt}}$
BERT
$o_{t,j}^{\mathrm{attr}}$
one-hot encoding
$e_{t,j}$
$e_{t,j}^{\mathrm{attr}}$ $e_{t,j}^{\mathrm{txt}}$ $e_{t,j}^{\mathrm{img}}$
x_t
e^g
$e_{t-1,0}$
$e_{t-1,1}$
e_{t-1}^{type}
e_{t-1}^{spec}
$e_{t,i}$
$e_{t,j}$
e^{end}
e_t^{type}
next action prediction
a_t^{type}
a_t^{spec}

Action Transformer Training

The action type prediction head over the 10 possible action types. It is trained using to maximise the log likelihood of the correct action type

$$\mathcal{L}_{\text{type}} = -\mathbf{E}_{a^{\text{type}}, x \in \mathcal{D}} \left[\log(p(a^{\text{type}} | h)) \right]$$

Action Transformer Training

We also project the hidden states of AcT to a space that we will perform contrastive learning. Compare the hidden state of the action type embedding with the hidden state of all UI elements.

$$q^{\text{type}} = f_{\text{target}}(h_t^{\text{type}}) \quad \text{and} \quad p^{\text{ui}} = f_{\text{target}}(h^{\text{ui}}) \qquad\qquad S = \frac{qp^T}{||q|| \cdot ||p||_r}\tau$$

Action Transformer Training

We also project the hidden states of AcT to a space that we will perform contrastive learning. Compare the hidden state of the action type embedding with the hidden state of all UI elements.

$$q^{\text{type}} = f_{\text{target}}(h_t^{\text{type}}) \quad \text{and} \quad p^{\text{ui}} = f_{\text{target}}(h^{\text{ui}}) \qquad\qquad S = \frac{qp^T}{\|q\| \cdot \|p\|_r}\tau$$

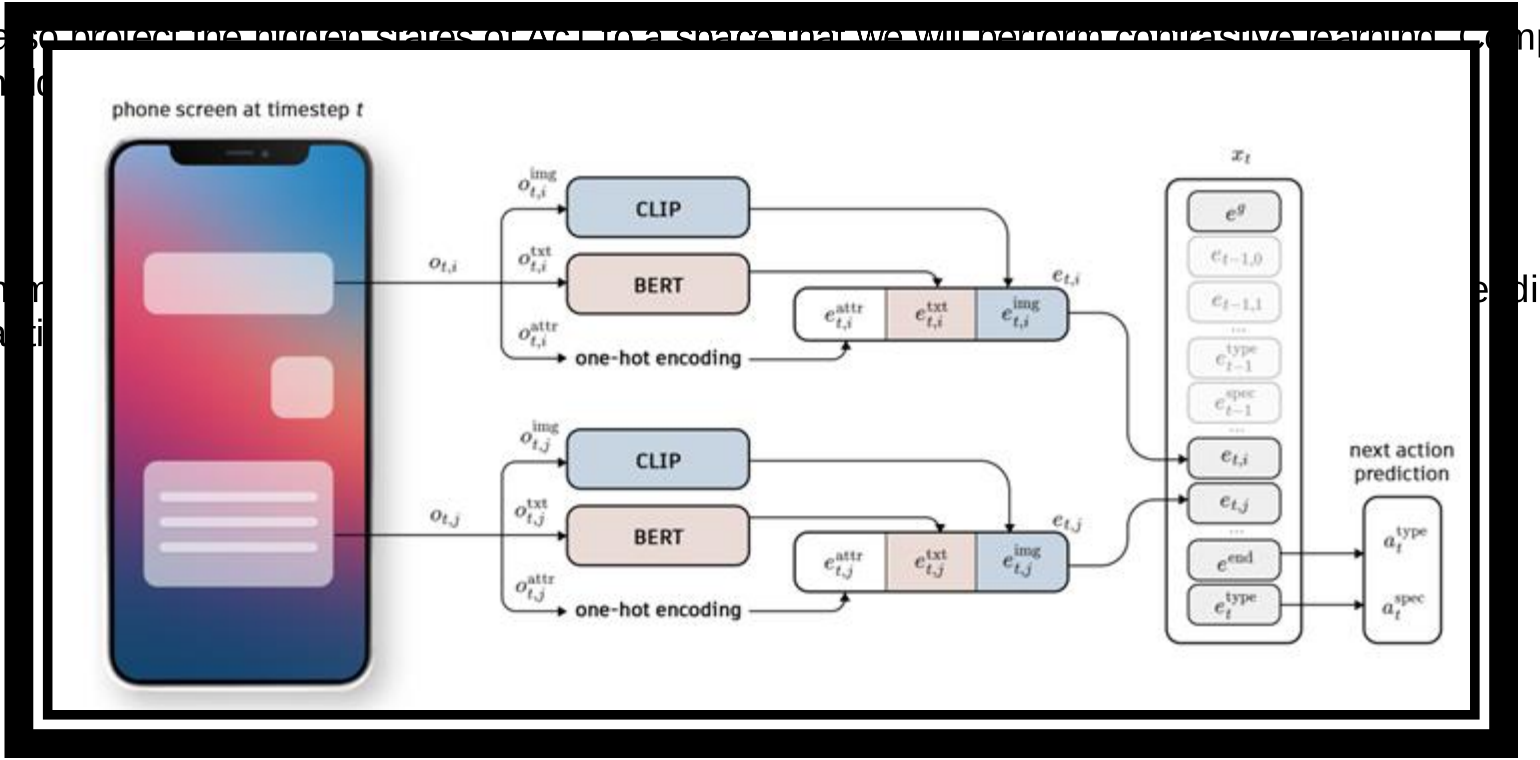
Then minimise the InfoNCE loss to project the correct target element embedding close to embedding of the action type

$$\mathcal{L}_{\text{elem}} = -\mathbf{E}\left[\log \frac{\exp(S_+)}{\sum_{i=1}^{K} \exp(S_i)}\right]$$

Action Transformer Training

We also project the hidden states of ACT to a space that we will perform contrastive learning. Compare the h

Then m … ding of the a



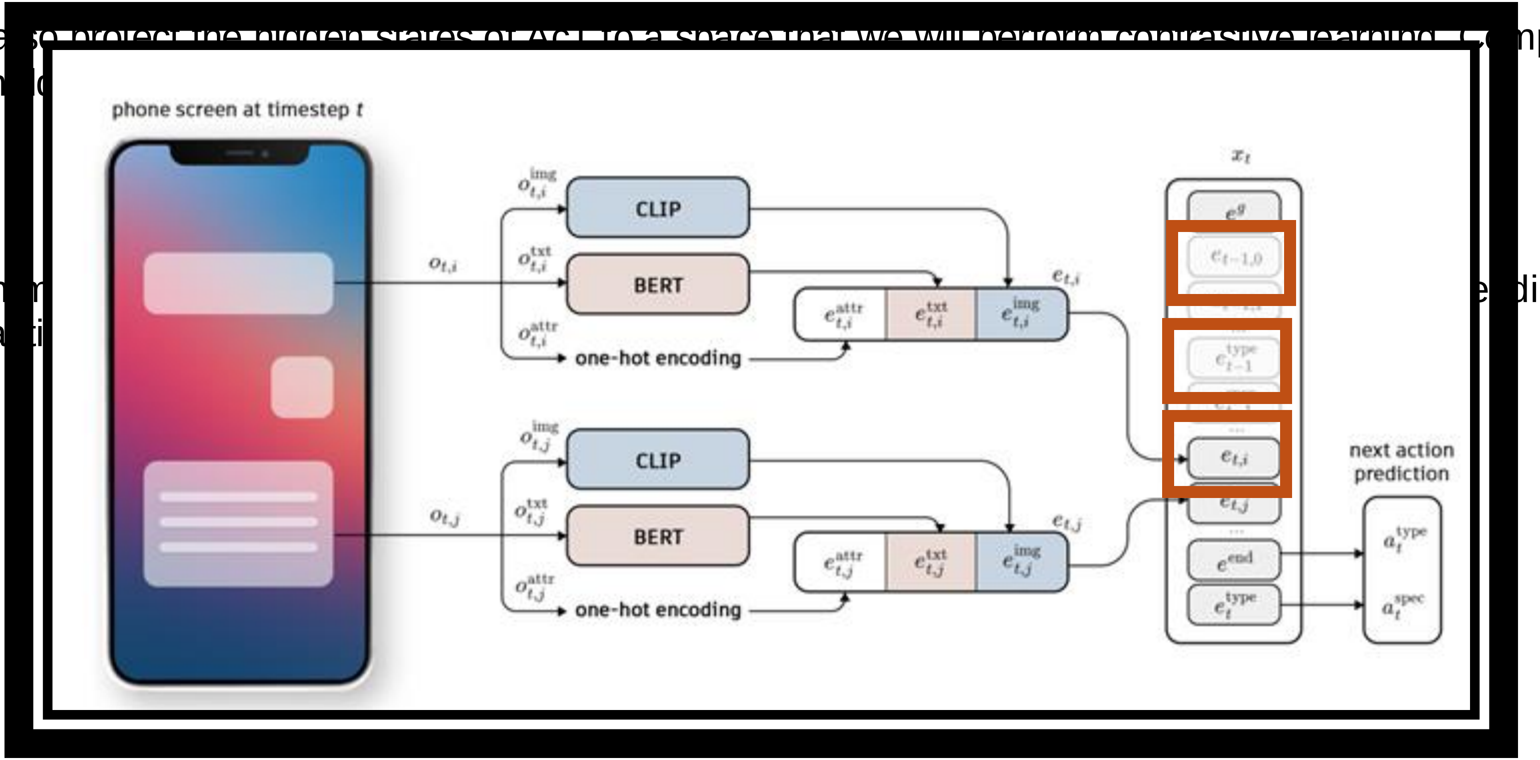
Action Transformer Training

We a … mpare

the h

Then … ding of

the a



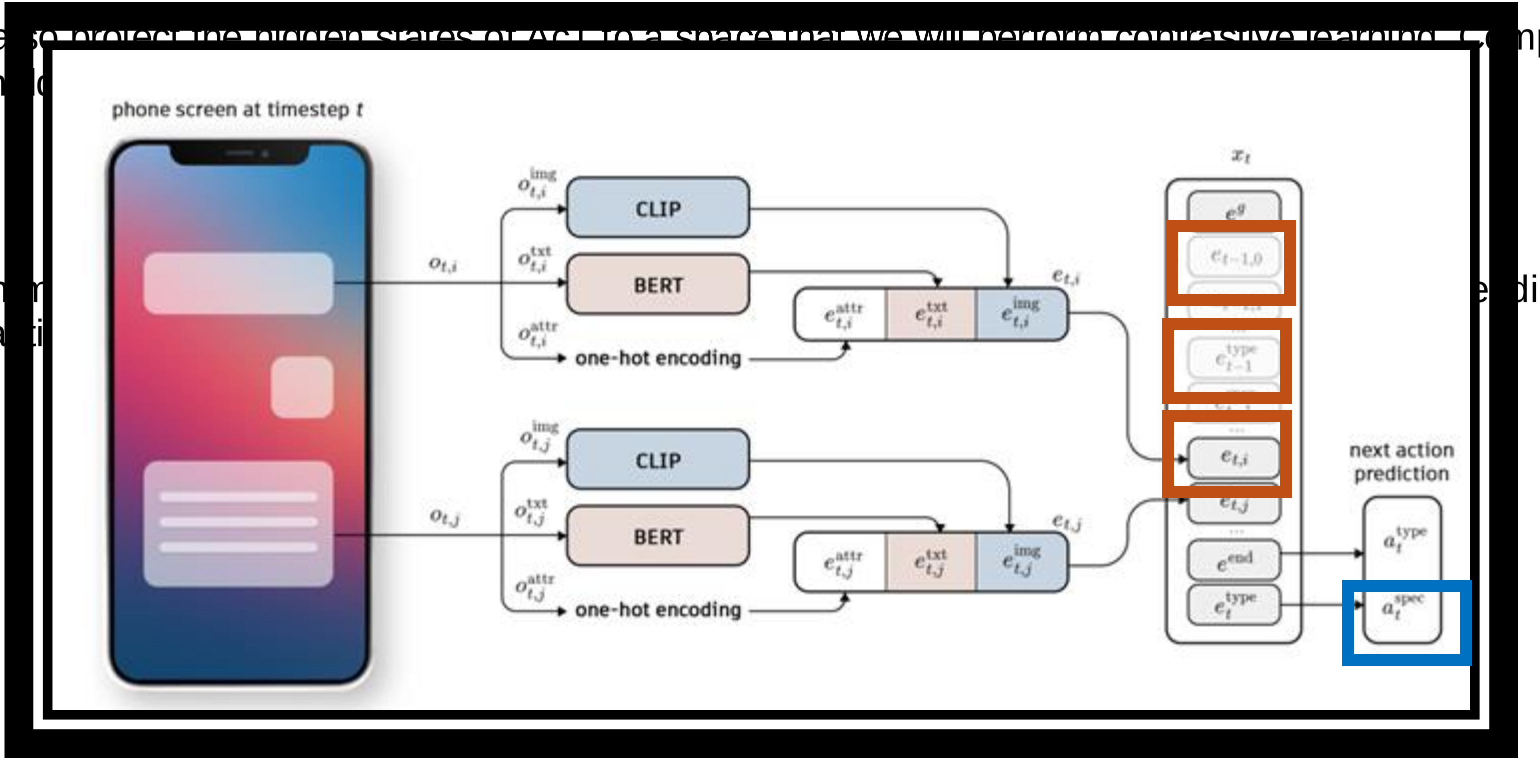
Action Transformer Training

We a [redacted] mpare

the h

Then ding of

the a

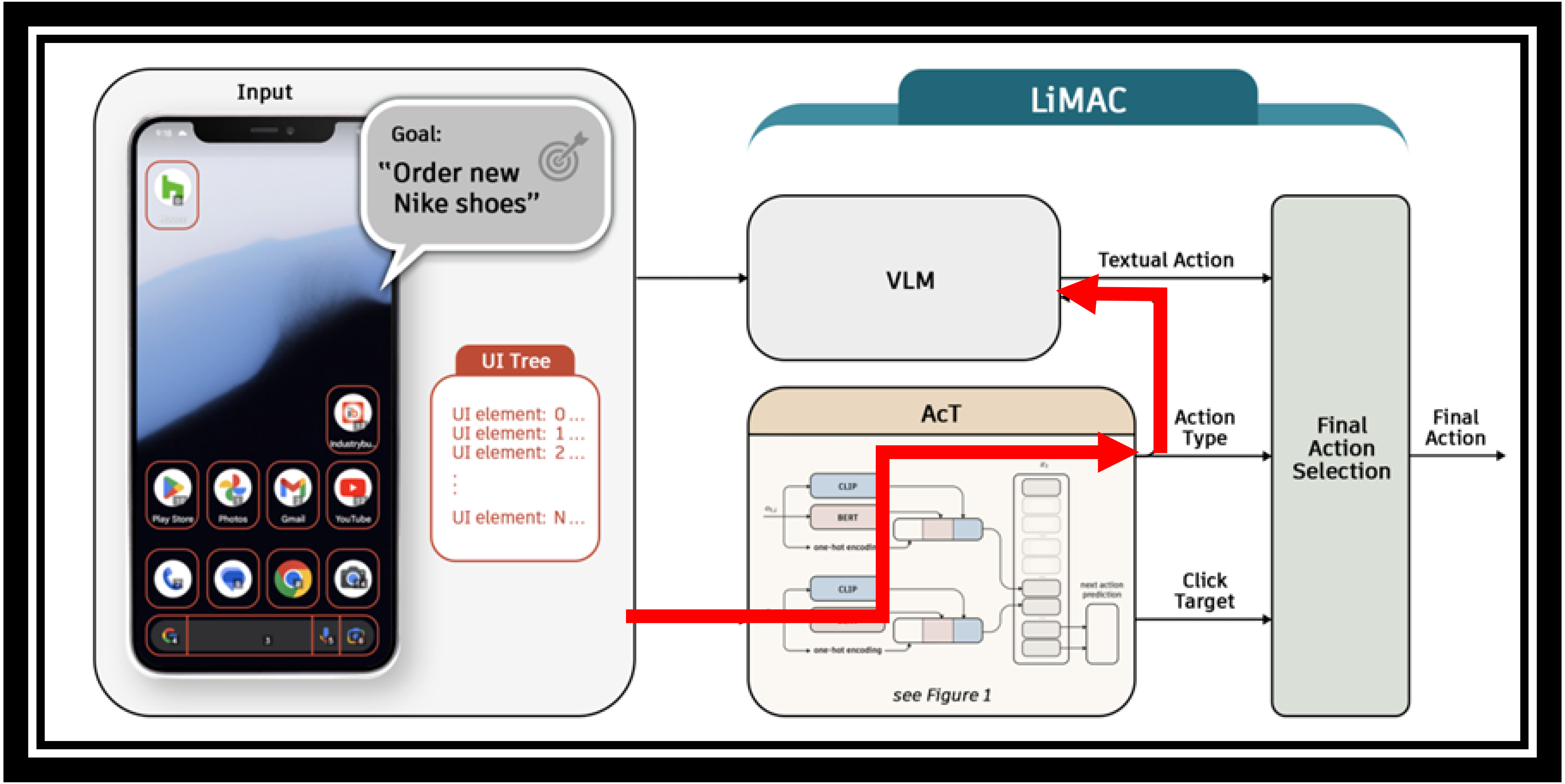


VLM fine-tuning

- LiMAC uses AcT to predict the action type and the target element of click actions.
- Actions that require text for the specifications are handled by a small VLM.
- The VLM receives as input the screenshot with bounding boxes around the UI elements with a corresponding number as well as the goal in natural language
- The VLM is fine-tuned using the training dataset in all actions
- The VLM is fine-tuned to maximise the log likelihood
- We fine-tune two different VLMs:

  Florence2 which is 820M parameters

  Qwen2-VL which is 2B parameters


Input
Goal:
"Order new Nike shoes"
UI Tree
UI element: 0 ...
UI element: 1 ...
UI element: 2 ...
UI element: N ...
LiMAC
VLM
Textual Action
AcT
Action Type
Click Target
Final Action Selection
Final Action
see Figure 1

Overall Accuracy Evaluation

Model	Size ↓	Avg Inf. Time (s) ↓	Overall ↑	
			AitW	AndCtrl
$\text{SeeAct}_{\text{choice}}$	unk	9.81	37.7	29.9
$\text{SeeAct}_{\text{ann}}$	unk	9.76	42.5	35.5
T3A	unk	4.87	26.9	53.1
M3A	unk	10.64	35.6	57.5
Florence2	820M	0.50	70.8	57.0
LiMAC with Florence2 (ours)	+520M	**0.34**	**72.2**	**63.1**
Qwen2-VL	2B	3.03	51.0	52.2
LiMAC with Qwen2-VL (ours)	+520M	0.63	70.9	62.5

Overall Accuracy Evaluation

Model	Size ↓	Avg Inf. Time (s)↓	Overall ↑	
			AitW	AndCtrl
$\text{SeeAct}_{\text{choice}}$	unk	9.81	37.7	29.9
$\text{SeeAct}_{\text{ann}}$	unk	9.76	42.5	35.5
T3A	unk	4.87	26.9	53.1
M3A	unk	10.64	35.6	57.5
Florence2	820M	0.50	70.8	57.0
LiMAC with Florence2 (ours)	+520M	**0.34**	**72.2**	**63.1**
Qwen2-VL	2B	3.03	51.0	52.2
LiMAC with Qwen2-VL (ours)	+520M	0.63	70.9	62.5

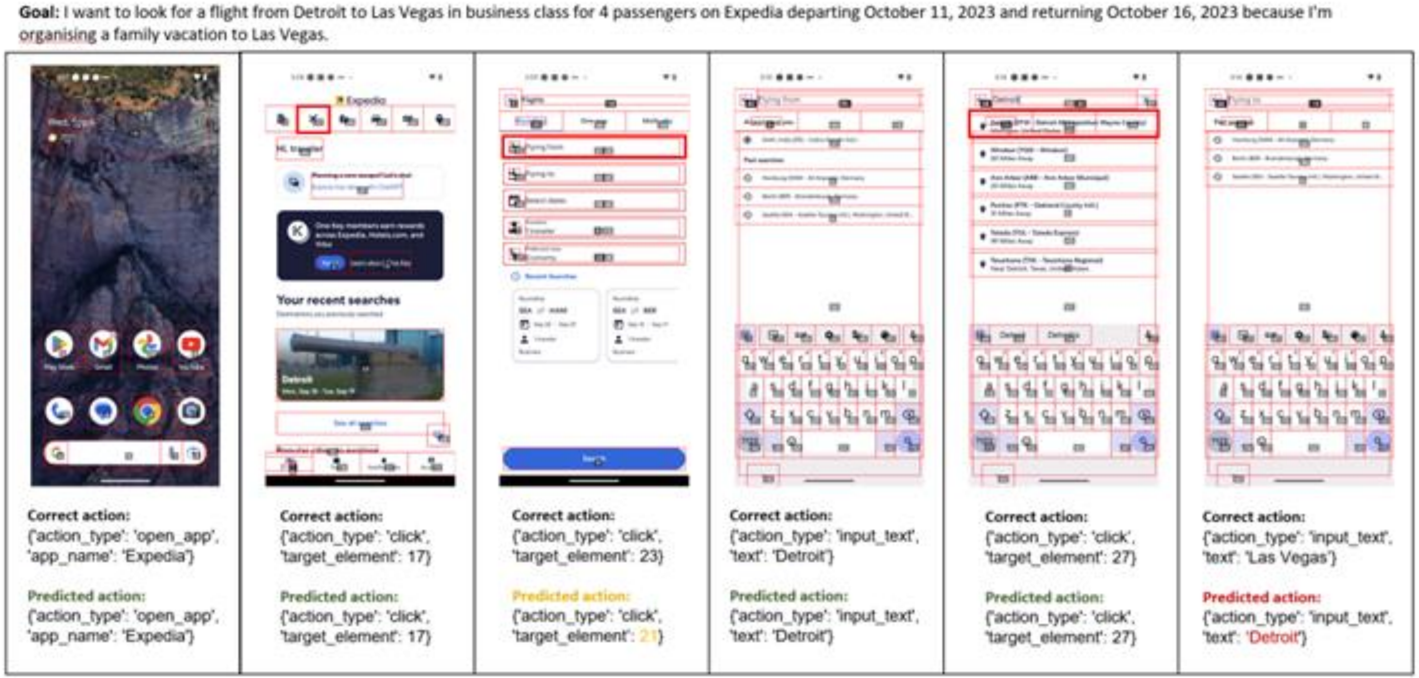
Ablation Study (Action-Type and Click-Target Accuracy)

Framework	Modules Used			Action Type		Click Target		Text	
	Type	Click	Text	AitW	AndCtrl	AitW	AndCtrl	AitW	AndCtrl
SeeAct only	$\text{SeeAct}_{\text{choice}}$	$\text{SeeAct}_{\text{choice}}$	$\text{SeeAct}_{\text{choice}}$	67.1	66.8	36.9	48.5	69.4	67.1
SeeAct only	$\text{SeeAct}_{\text{ann}}$	$\text{SeeAct}_{\text{ann}}$	$\text{SeeAct}_{\text{ann}}$	68.2	66.8	44.7	55.7	66.0	61.8
T3A only	T3A	T3A	T3A	56.2	67.7	33.5	71.1	66.5	**78.4**
M3A only	M3A	M3A	M3A	63.8	69.8	48.3	**77.1**	67.3	74.3
Qwen only	Qwen2-VL	Qwen2-VL	Qwen2-VL	81.7	70.7	53.2	55.2	70.5	75.7
LiMAC (ours)	AcT	Qwen2-VL	Qwen2-VL	**86.9**	**82.3**	53.2	55.2	70.5	75.7
LiMAC (ours)	AcT	AcT	Qwen2-VL	**86.9**	**82.3**	**77.4**	65.4	70.5	75.7
Florence only	Florence2	Florence2	Florence2	86.4	79.6	76.2	62.0	**84.2**	77.5
LiMAC (ours)	AcT	Florence2	Florence2	**86.9**	**82.3**	76.2	62.0	**84.2**	77.5
LiMAC (ours)	AcT	AcT	Florence2	**86.9**	**82.3**	**77.4**	65.4	**84.2**	77.5

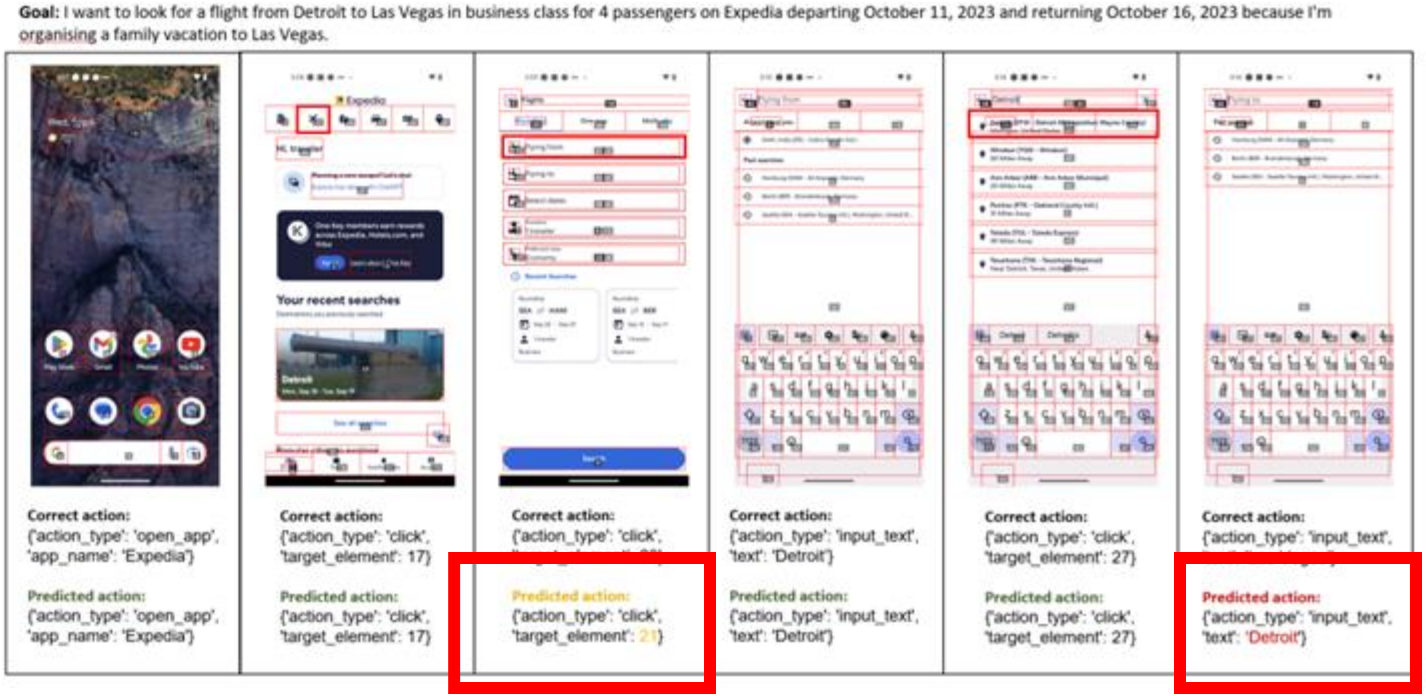
Ablation Study (Action-Type and Click-Target Accuracy)

Framework	Modules Used			Action Type		Click Target		Text	
	Type	Click	Text	AitW	AndCtrl	AitW	AndCtrl	AitW	AndCtrl
SeeAct only	$\text{SeeAct}_{\text{choice}}$	$\text{SeeAct}_{\text{choice}}$	$\text{SeeAct}_{\text{choice}}$	67.1	66.8	36.9	48.5	69.4	67.1
SeeAct only	$\text{SeeAct}_{\text{ann}}$	$\text{SeeAct}_{\text{ann}}$	$\text{SeeAct}_{\text{ann}}$	68.2	66.8	44.7	55.7	66.0	61.8
T3A only	T3A	T3A	T3A	56.2	67.7	33.5	71.1	66.5	**78.4**
M3A only	M3A	M3A	M3A	63.8	69.8	48.3	**77.1**	67.3	74.3
Qwen only	Qwen2-VL	Qwen2-VL	Qwen2-VL	81.7	70.7	53.2	55.2	70.5	75.7
LiMAC (ours)	AcT	Qwen2-VL	Qwen2-VL	**86.9**	**82.3**	53.2	55.2	70.5	75.7
LiMAC (ours)	AcT	AcT	Qwen2-VL	**86.9**	**82.3**	**77.4**	65.4	70.5	75.7
Florence only	Florence2	Florence2	Florence2	86.4	79.6	76.2	62.0	**84.2**	77.5
LiMAC (ours)	AcT	Florence2	Florence2	**86.9**	**82.3**	76.2	62.0	**84.2**	77.5
LiMAC (ours)	AcT	AcT	Florence2	**86.9**	**82.3**	**77.4**	65.4	**84.2**	77.5

Case Study of an Episode



Case Study of an Episode



Multi-Agent RL – Resources

The MIT Press

Multi-Agent Reinforcement Learning: An Introduction

Find it online for free! www.marl-book.com

Code repo: github.com/uoe-agents

- 25 active code repos
- Extended PyMARL + blog post

Book codebase: github.com/marl-book/codebase