

# Structured Losses

## Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning

Junhyuk Oh, Satinder Singh, Honglak Lee, Pushmeet Kohli

Oh et al. 2017 <https://arxiv.org/abs/1706.05064>

**Presented by Belén Saldías**  
belen@mit.edu

Friday, November 6, 2020

# Outline

- |                              |               |             |
|------------------------------|---------------|-------------|
| 1. Paper: Oh et al. 2017     | 11:35 - 12:05 | (~ 30 mins) |
| 2. Breakout rooms discussion | 12:05 - 12:20 | (~ 15 mins) |
| 3. Class discussion          | 12:20 - 12:30 | (~ 10 mins) |

# Zero-Shot Task Generalization with Multi-Task Deep Reinforcement Learning

Oh et al. 2017

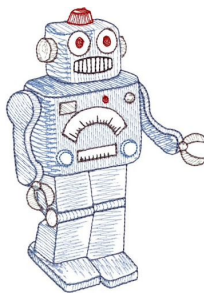
- Problem set up
- Approach and technical contributions
- Related work
- Learning a Parameterized Skill
- Learning to Execute Sequential Instructions
- Conclusions & Takeaways
- Discussion

**Feel free to raise your blue-Zoom  
hand if you want to add something  
as the presentation goes!**

# Motivation: Zero-shot task generalization

**Problem:** It is infeasible to train a household robot to do every possible combinations of instructions.

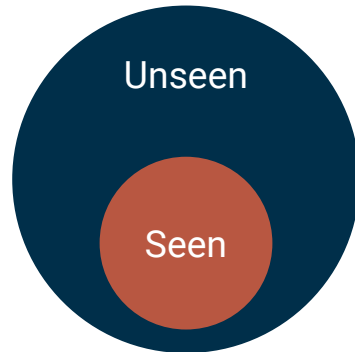
**Goal:** Train the agent on a small set of tasks such that it can generalize over a larger set of tasks without additional training.



## Training

1. Go to the kitchen
2. Wash dishes
3. Empty the trash can
4. Go to the bedroom

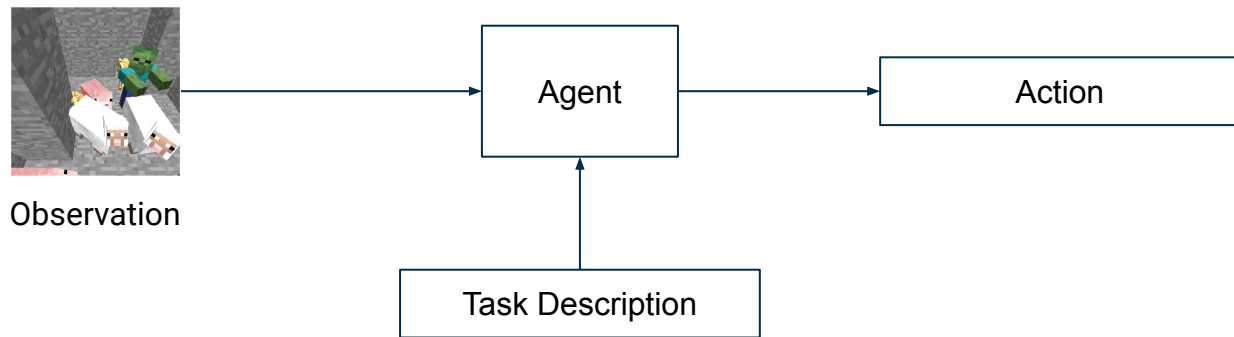
Task  
space



# Motivation: Multi-task Deep Reinforcement Learning (RL)

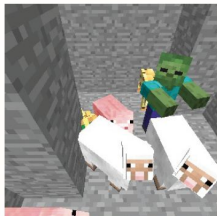
The agent is required to:

- Perform many different tasks depending on the given task description.
- Generalize over unseen task descriptions.

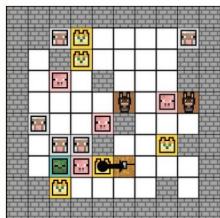
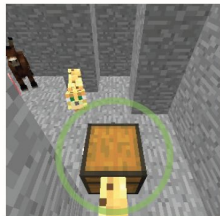
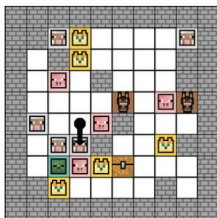


# Problem set up

First-person-view  
(Observation)



Top-down-view  
(Not available)



## Training

1. Visit pig
2. Pick up 3 sheep
3. Transform greenbot
4. Pick up horse

## Testing

1. Pick up pig
2. Visit sheep
3. Transform cat
4. Transform 3 sheep
5. Pick up greenbot
6. Pick up 2 pig
7. Transform 2 sheep
8. Transform 2 cat
- ...

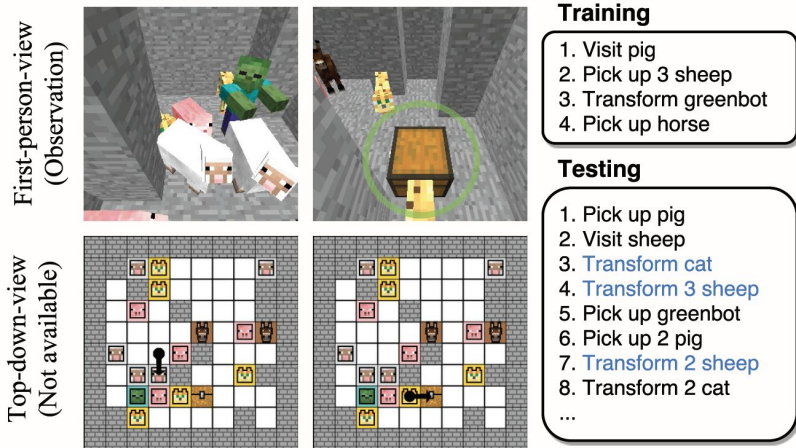
## Task:

Instruction execution: an agent's task is to **execute a given list of instructions** described by a **simple form of natural language** while dealing with **unexpected events**.

## Assumption:

Each **instruction** can be executed by performing one or more **high-level subtask** in sequence.

# Problem set up



## Task:

Instruction execution: an agent's task is to **execute a given list of instructions** described by a **simple form of natural language** while dealing with **unexpected events**.

## Assumption:

Each **instruction** can be executed by performing one or more **high-level subtask** in sequence.

## Challenges:

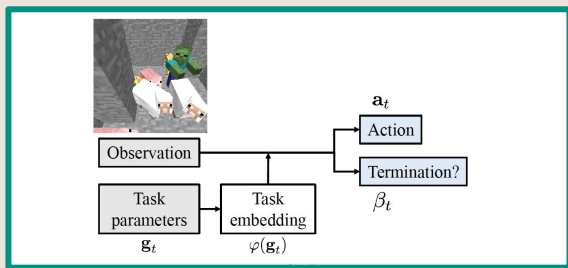
- Generalization
  - Unseen subtasks (skill learning stage)
  - Longer sequences of instructions
- Delayed reward (subtask updater)
- Interruptions (bonus or emergencies)
- Memory (loop tasks)

# Approach and technical contributions

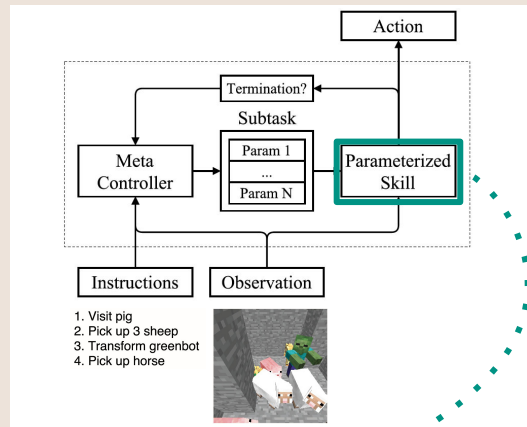
The learning problem is divided in two stages

- 1) Learning **parameterized skills** to perform subtasks and generalize to unseen subtasks.

subtask := several disentangled parameters



- 2) Learning to execute instructions using the learned skills.



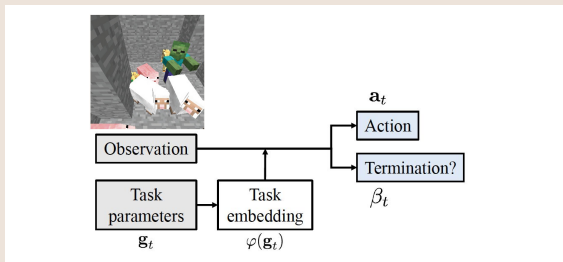


# Approach and technical contributions

The learning problem is divided in two stages

- 1) Learning **parameterized skills** to perform subtasks and generalize to unseen subtasks.

subtask := several disentangled parameters



## How to generalize?

New objective function that encourages **making analogies between similar subtasks** so that the manifold of the subtasks spaces can be learned without experiencing all subtasks.

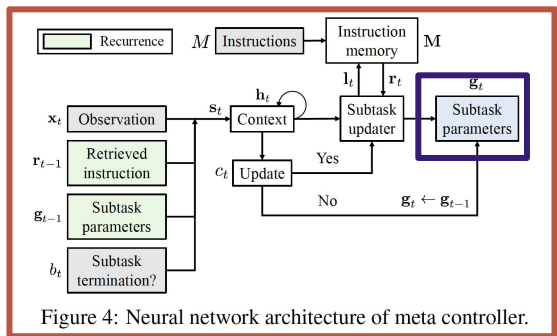
The authors show that the analogy-making objective can generalize successfully.

# Approach and technical contributions

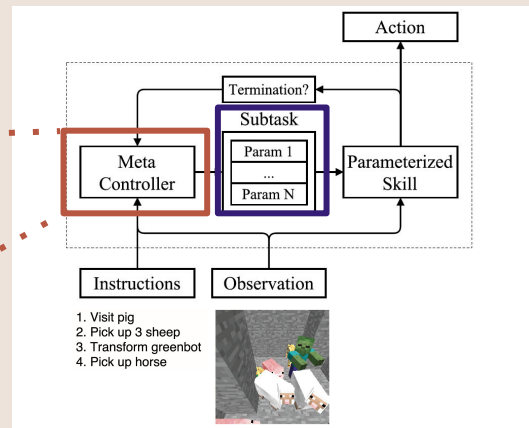
The learning problem is divided in two stages

## How to generalize?

The **meta controller**'s ability to learn when to update a subtask plays a key role in solving the overall problem.



## 2) Learning to execute instructions using the learned skills.



# Related work

## Hierarchical RL

- Much of previous work assumed an optimal sequence of subtasks fixed during evaluation. Also using meta *meta controller* and a set *low-level controllers for subtasks*.
- Makes it hard to evaluate the **agent's ability to solve previously unseen sequential tasks in a zero-shot fashion** unless the agent is trained on the new tasks.
- Different to previous work, in this work instructions are a description of the tasks, where the agent needs to learn to use these descriptions to maximize reward.

## Hierarchical Deep RL

- Most of the recent work on hierarchical RL and deep learning build an **open-loop policy** at the high-level controller that waits until the previous subtask is finished to trigger the next subtask.
- This open-loop approach is not able to handle interruptions, while **the authors proposed architecture can switch its subtask at any time.**

# Related work

## Zero-Shot Task Generalization

- Some previous work aimed at generalization by mapping task descriptions to policies or using sub-networks that are shared across tasks.
- Andreas et al. (2016) proposes a framework to generalize over new sequence of learned tasks.
- The authors propose a flexible **metric learning method (i.e., analogy-making)** that can be applied to various generalization scenarios.
- This work aims to generalize to both to unseen tasks and unseen sequences of them.

## Instruction execution

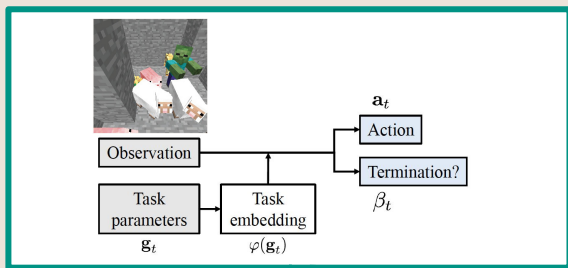
- Some work has focused on using natural language understanding to map instructions to actions.
- This work focuses on generalization to sequences of instructions without any supervision for language understanding or for actions.
- Branan et al. (2009) tackles a similar problem but with only a single instruction at a time, while the authors' agent works on aligning a list of instructions and internal state.

# Approach

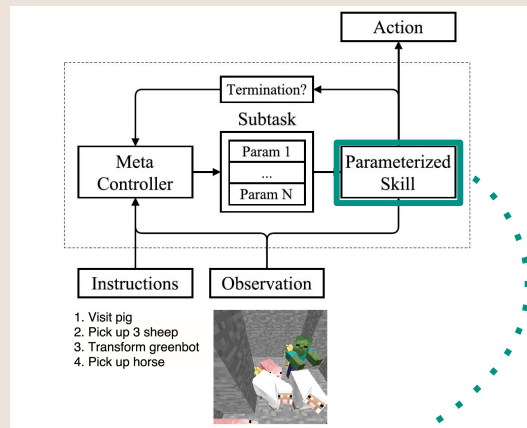
The learning problem is divided in two stages

- 1) Learning **parameterized skills** to perform subtasks and generalize to unseen subtasks.

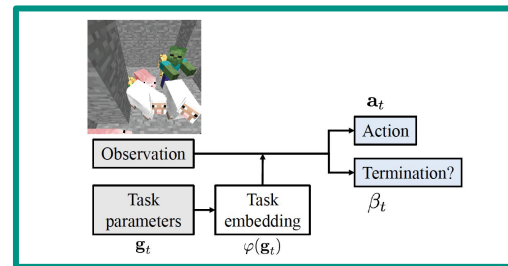
subtask := several disentangled parameters



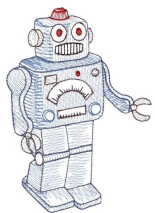
- 2) Learning to execute instructions using the learned skills.



# 1) Learning a Parameterized Skill



Object-independent scenario



**Training**

Pick up (📦)  
Throw (⚽)

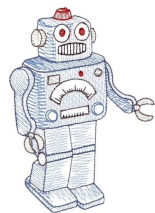
**Testing**

Pick up (🎾)

To generalize, the agent assumes:

- Semantics of each parameter are consistent.
- Required knowledge: "Pick up ⚽ as you pick up 📦."

Object-dependent scenario



**Training**

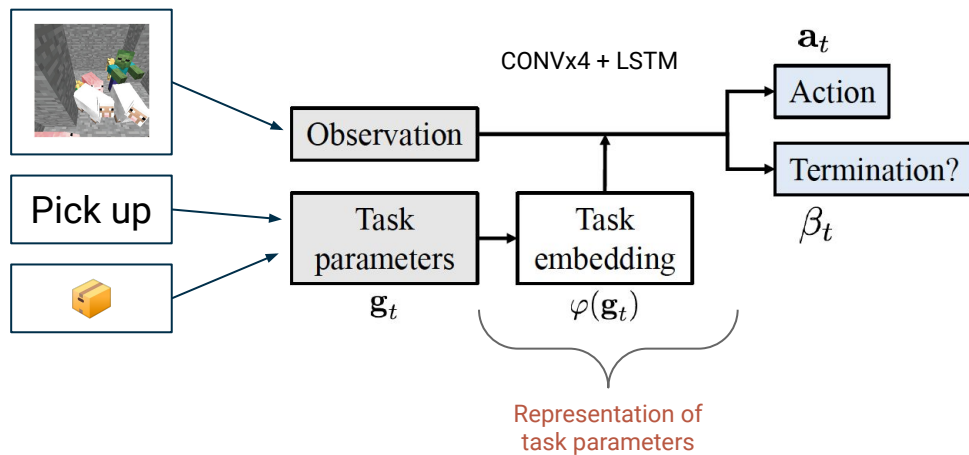
Interact (🍏) = eat  
Interact (⚽) = throw

**Testing**

Interact (🍷) = eat

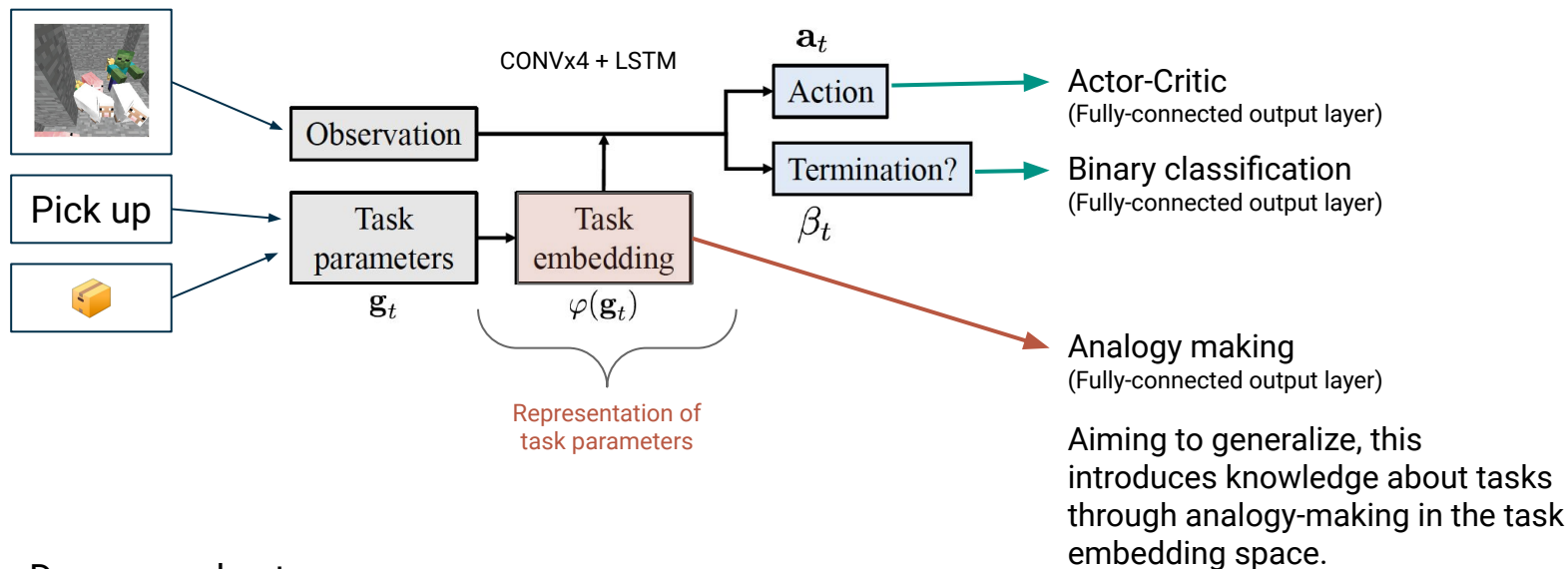
- Semantics of a task depend on a combination of parameters (e.g., target object).
- Impossible to generalize over unseen combinations without any prior knowledge.
- Required knowledge: "Interact with 🍷 as you interact with 🍏."

# 1) Learning a Parameterized Skill



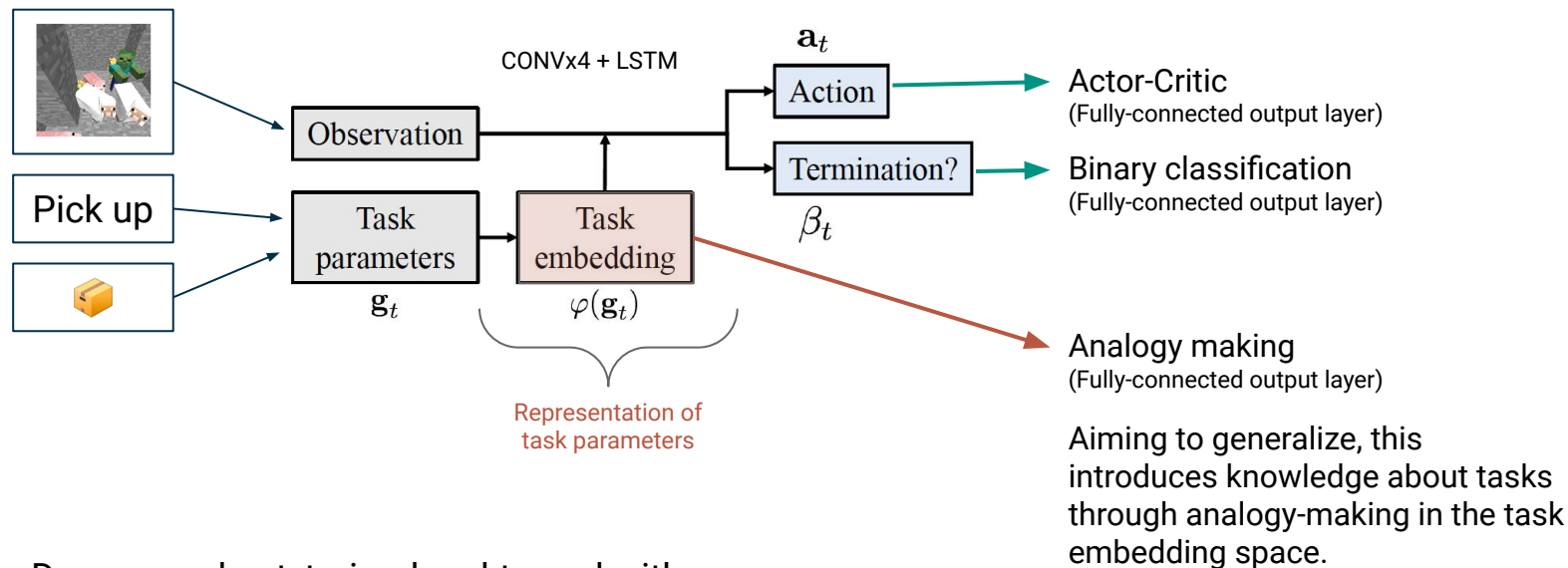
Deep neural net

# 1) Learning a Parameterized Skill





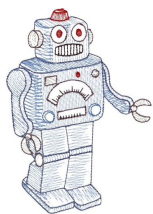
# 1) Learning a Parameterized Skill



Deep neural net, trained end-to-end with these three objectives.

## 1.1) Learning to Generalize by Analogy-Making

## Object-independent scenario


$$[\text{Visit}, X] : [\text{Visit}, Y] :: [\text{Pick up}, X] : [\text{Pick up}, Y]$$

$[\text{Visit}, X] \xrightarrow{\text{difference}} [\text{Pick up}, X]$

$[Visit, Y] \xrightarrow{\text{difference}} [Pick\ up, Y]$

## Constraints in embedding space

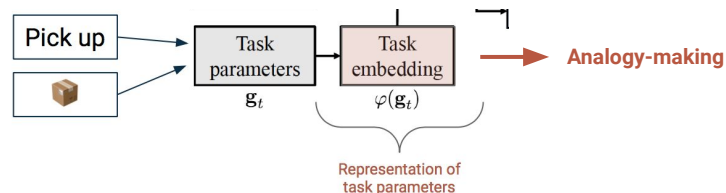
$$\Delta(\mathbf{g}_A, \mathbf{g}_B) = \varphi(\mathbf{g}_A) - \varphi(\mathbf{g}_B)$$

$$\|\Delta(\mathbf{g}_A, \mathbf{g}_B) - \Delta(\mathbf{g}_C, \mathbf{g}_D)\| \approx 0 \quad \text{if } \mathbf{g}_A : \mathbf{g}_B :: \mathbf{g}_C : \mathbf{g}_D$$

$$\|\Delta(\mathbf{g}_A, \mathbf{g}_B) - \Delta(\mathbf{g}_C, \mathbf{g}_D)\| \geq \tau_{dis} \quad \text{if } \mathbf{g}_A : \mathbf{g}_B \neq \mathbf{g}_C : \mathbf{g}_D$$

$$\|\Delta(\mathbf{g}_A, \mathbf{g}_B)\| \geq \tau_{diff} \quad \text{if } \mathbf{g}_A \neq \mathbf{g}_B,$$

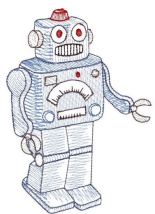
Goal: learn correspondence between tasks.



Acquire knowledge about the relationship between different task parameters when learning the task embedding.

## 1.1) Learning to Generalize by Analogy-Making

## Object-independent scenario


$$[\text{Visit}, X] : [\text{Visit}, Y] :: [\text{Pick up}, X] : [\text{Pick up}, Y]$$

$[\text{Visit}, X] \xrightarrow{\text{difference}} [\text{Pick up}, X]$

$[Visit, Y] \xrightarrow{\text{difference}} [Pick\ up, Y]$

## Constraints in embedding space

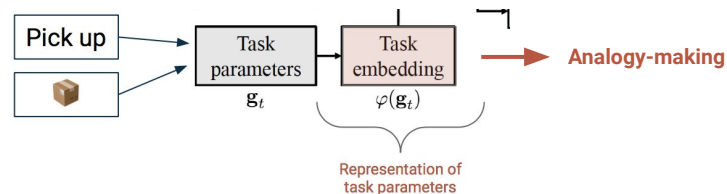
$$\Delta(\mathbf{g}_A, \mathbf{g}_B) = \varphi(\mathbf{g}_A) - \varphi(\mathbf{g}_B)$$

$$\|\Delta(\mathbf{g}_A, \mathbf{g}_B) - \Delta(\mathbf{g}_C, \mathbf{g}_D)\| \approx 0 \quad \text{if } \mathbf{g}_A : \mathbf{g}_B :: \mathbf{g}_C : \mathbf{g}_D$$

$$\|\Delta(\mathbf{g}_A, \mathbf{g}_B) - \Delta(\mathbf{g}_C, \mathbf{g}_D)\| \geq \tau_{dis} \quad \text{if } \mathbf{g}_A : \mathbf{g}_B \neq \mathbf{g}_C : \mathbf{g}_D$$

$$\|\Delta(\mathbf{g}_A, \mathbf{g}_B)\| \geq \tau_{diff} \quad \text{if } \mathbf{g}_A \neq \mathbf{g}_B,$$

Goal: learn correspondence between tasks.

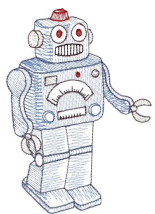


**Analogy-making** (similar to Mikolov et al. (2013)).

Prevent trivial solutions and learn differences between tasks.

## 1.1) Learning to Generalize by Analogy-Making

## Object-independent scenario


$$[\text{Visit}, X] : [\text{Visit}, Y] :: [\text{Pick up}, X] : [\text{Pick up}, Y]$$

$[\text{Visit}, X] \xrightarrow{\text{difference}} [\text{Pick up}, X]$

$[Visit, Y] \xrightarrow{\text{difference}} [Pick\ up, Y]$

## Constraints in embedding space

$$\Delta(\mathbf{g}_A, \mathbf{g}_B) = \varphi(\mathbf{g}_A) - \varphi(\mathbf{g}_B)$$

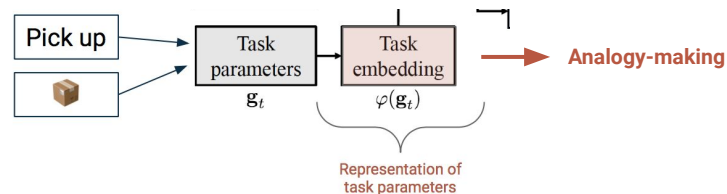
$$\mathcal{L}_{sim} = \mathbb{E}_{\mathbf{g}_{A...D} \sim \mathcal{G}_{sim}} [\|\Delta(\mathbf{g}_A, \mathbf{g}_B) - \Delta(\mathbf{g}_C, \mathbf{g}_D)\|^2]$$

$$\mathcal{L}_{dis} = \mathbb{E}_{\mathbf{g}_{A...D} \sim \mathcal{G}_{dis}} \left[ (\tau_{dis} - \|\Delta(\mathbf{g}_A, \mathbf{g}_B) - \Delta(\mathbf{g}_C, \mathbf{g}_D)\|_+)^2 \right]$$

$$\mathcal{L}_{diff} = \mathbb{E}_{\mathbf{g}_{A,B} \sim \mathcal{G}_{diff}} \left[ (\tau_{diff} - \|\Delta(\mathbf{g}_A, \mathbf{g}_B)\|_+)^2 \right],$$

$$\mathcal{L}_{AM} = \mathcal{L}_{sim} + \rho_1 \mathcal{L}_{dis} + \rho_2 \mathcal{L}_{diff}$$

Goal: learn correspondence between tasks.

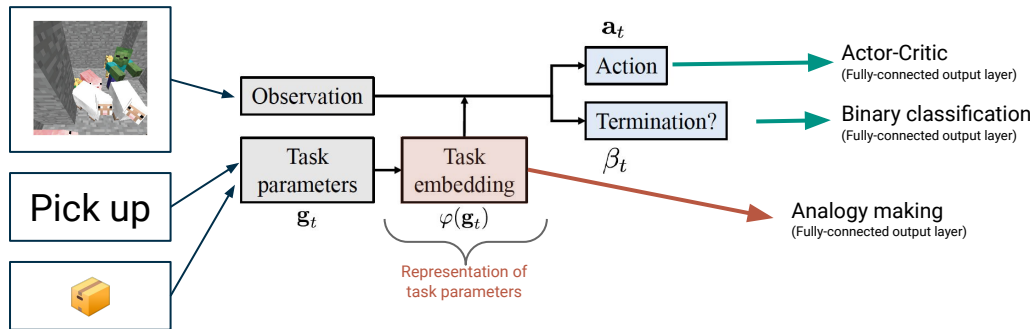


**Analogy-making** (similar to Mikolov et al. (2013)).

Prevent trivial solutions and learn differences between tasks.

Weighted sum of these three restrictions is added as a regularizer.

# 1) Learning a Parameterized Skill



The final update rule for the parameterized skill is:

$$\Delta\phi \propto -(\nabla_{\phi}\mathcal{L}_{RL} + \xi\nabla_{\phi}\mathcal{L}_{AM}), \quad (7)$$

$$\nabla_{\phi}\mathcal{L}_{RL} = \mathbb{E}_{g \sim \mathcal{U}} \left[ \mathbb{E}_{s \sim \pi_{\phi}^g} \left[ -\nabla_{\phi} \log \pi_{\phi}(a_t | s_t, g) \hat{A}_t^{(\gamma, \lambda)} + \alpha \nabla_{\phi} \mathcal{L}_{term} \right] \right],$$

Fine-tune multi-task  
policy

cross-entropy loss for  
termination prediction

analogy-making regularizer

# 1.1) Learning to Generalize by Analogy-Making

## Results

Scenario
Independent
Object-dependent
Inter/Extrapolation

The semantics of the tasks are consistent across all types of target objects. **Generalize to unseen configuration of task parameters.**

Two groups: Group A and B. Given "interact with" action, Group A should be picked up, whereas Group B should be transformed. **To generalize to unseen objects, the agent needs to learn an embedding for the group.**

A task is defined by: action, object, and number. Repeat the same subtask for a given number of times. Trained in all actions and objects, but not all numbers. **The agent should generalize over unseen numbers.**

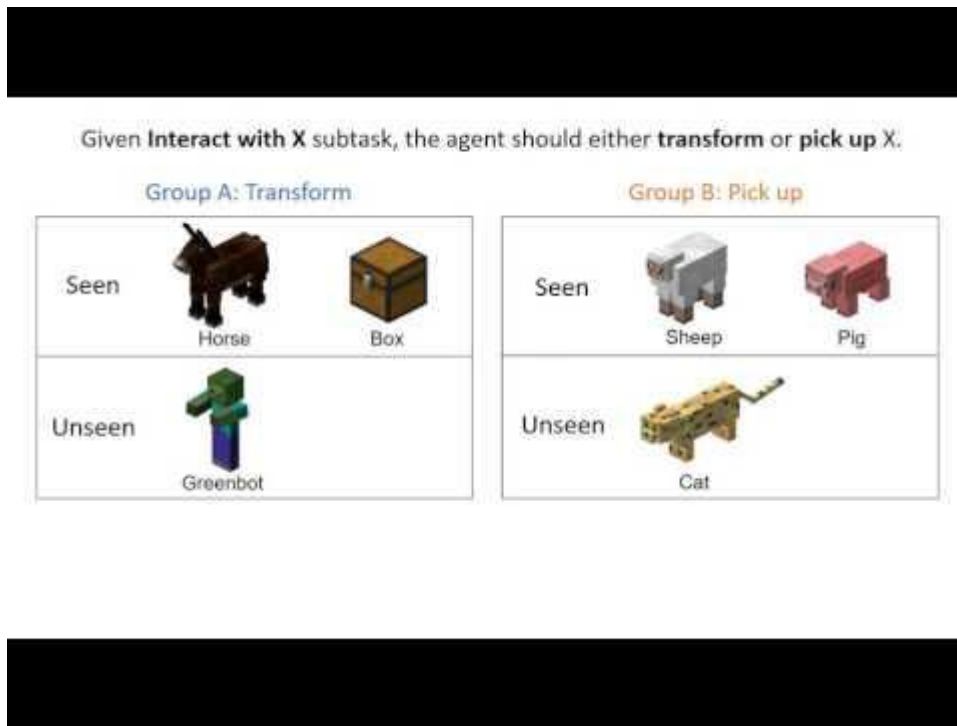
Sets of parameterized tasks

# 1.1) Learning to Generalize by Analogy-Making

Environment

Implementation details

- Curriculum training
- Actor-critic (parameters updated after 8 episodes).



# 1) Learning to Generalize by Analogy-Making

## Results

Scenario	Analogy	Train
Independent	×	<b>0.3</b> (99.8%)
	✓	<b>0.3</b> (99.8%)
Object-dependent	×	<b>0.3</b> (99.7%)
	✓	<b>0.3</b> (99.8%)
Inter/Extrapolation	×	<b>-0.7</b> (97.5%)
	✓	<b>-0.7</b> (97.5%)

Table 1: Performance on parameterized tasks. Each entry shows ‘Average reward (Success rate)’. We assume an episode is successful only if the agent successfully finishes the task and its termination predictions are correct throughout the whole episode.



# 1) Learning to Generalize by Analogy-Making

## Results

Scenario	Analogy	Train	Unseen
Independent	×	<b>0.3</b> (99.8%)	-3.7 (34.8%)
	✓	<b>0.3</b> (99.8%)	<b>0.3</b> (99.5%)
Object-dependent	×	<b>0.3</b> (99.7%)	-5.0 (2.2%)
	✓	<b>0.3</b> (99.8%)	<b>0.3</b> (99.7%)
Inter/Extrapolation	×	<b>-0.7</b> (97.5%)	-2.2 (24.9%)
	✓	<b>-0.7</b> (97.5%)	<b>-1.7</b> (94.5%)

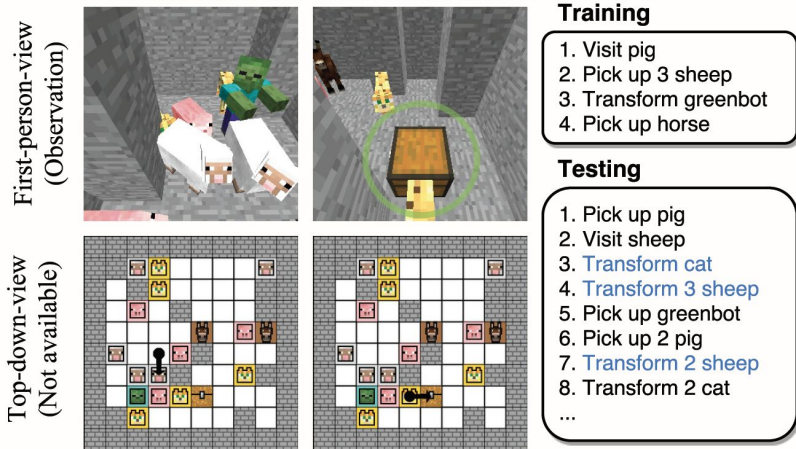
Table 1: Performance on parameterized tasks. Each entry shows ‘Average reward (Success rate)’. We assume an episode is successful only if the agent successfully finishes the task and its termination predictions are correct throughout the whole episode.

# 1) Learning to Generalize by Analogy-Making

## Takeaways

- When learning a representation of task parameters, it is possible to inject prior knowledge in the form of the analogy-making objective.
- Analogy-making, in this particular scenario, was crucial for generalization to unseen task parameters depending on semantics or context without needing to experience them.

# Problem set up



Task:

Instruction execution: an agent's task is to **execute a given list of instructions** described by a **simple form of natural language** while dealing with **unexpected events**.

Assumption:

Each **instruction** can be executed by performing one or more **high-level subtask** in sequence.

Challenges:

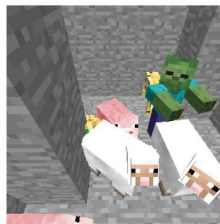
- Generalization
  - ~~Unseen subtasks (skill learning stage)~~
  - Longer sequences of instructions
- Delayed reward (subtask updater)
- Interruptions (bonus or emergencies)
- Memory (loop tasks)

## 2) Learning to execute instructions

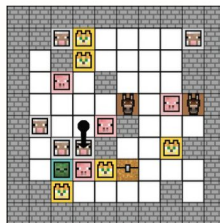
The agent needs to:

1. Execute a sequence of natural language instructions.  
Read one instruction at a time (pointer).  
Detect when the current instruction is finished.  
Memory (keep track of progress – counts)

First-person-view  
(Observation)



Top-down-view  
(Not available)



### Training

1. Visit pig
2. Pick up 3 sheep
3. Transform greenbot
4. Pick up horse

### Testing

1. Pick up pig
2. Visit sheep
3. Transform cat
4. Transform 3 sheep
5. Pick up greenbot
6. Pick up 2 pig
7. Transform 2 sheep
8. Transform 2 cat
- ...

## 2) Learning to execute instructions


The agent needs to:

1. Execute a sequence of natural language instructions.
  - Read one instruction at a time (pointer).
  - Detect when the current instruction is finished.
  - Memory (keep track of progress – counts)
2. Handle unexpected events (e.g., bonus or low battery).
  - Interrupt ongoing subtasks

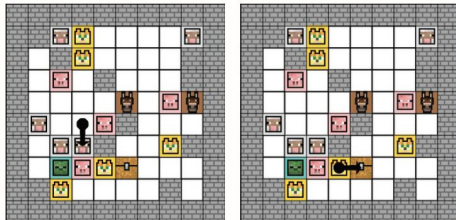
Assume:

- Already trained parameterized skills.

First-person-view  
(Observation)



Top-down-view  
(Not available)



**Training**

1. Visit pig
2. Pick up 3 sheep
3. Transform greenbot
4. Pick up horse

**Testing**

1. Pick up pig
2. Visit sheep
3. Transform cat
4. Transform 3 sheep
5. Pick up greenbot
6. Pick up 2 pig
7. Transform 2 sheep
8. Transform 2 cat
- ...

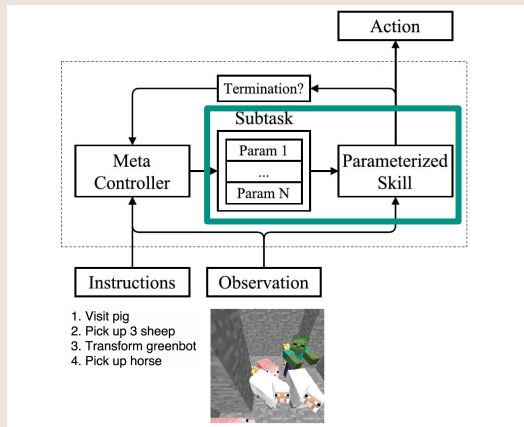
## 2) Learning to execute instructions

The learning problem is divided in two stages, stage 2:

### How to generalize?

The **meta controller**'s ability to learn when to update a subtask plays a key role in solving the overall problem.

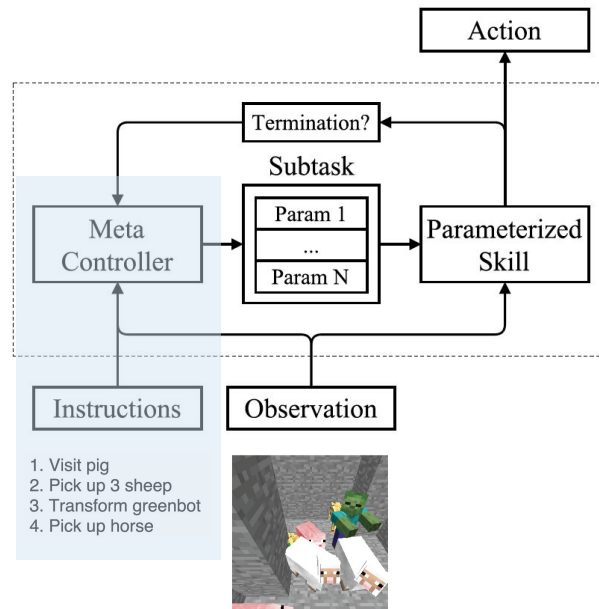
### 2) Learning to execute instructions using the learned skills.



## 2) Learning to execute instructions

### Architecture

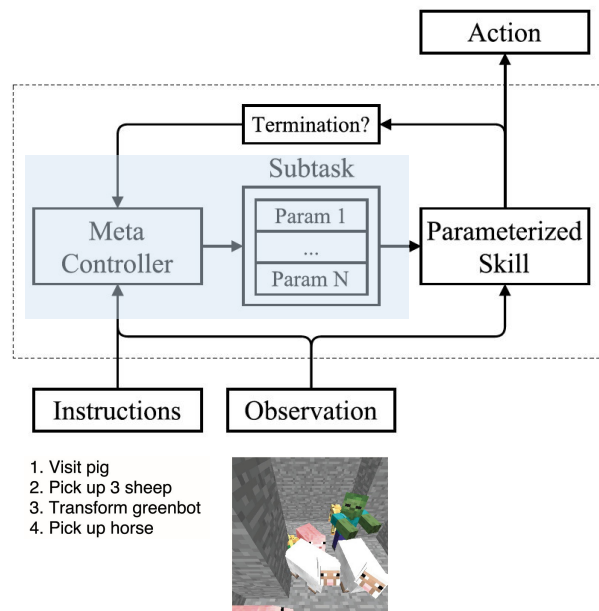
**Meta Controller:** reads instructions and



## 2) Learning to execute instructions

### Architecture

**Meta Controller:** reads instructions and passes subtask parameters to the parameterized skill.



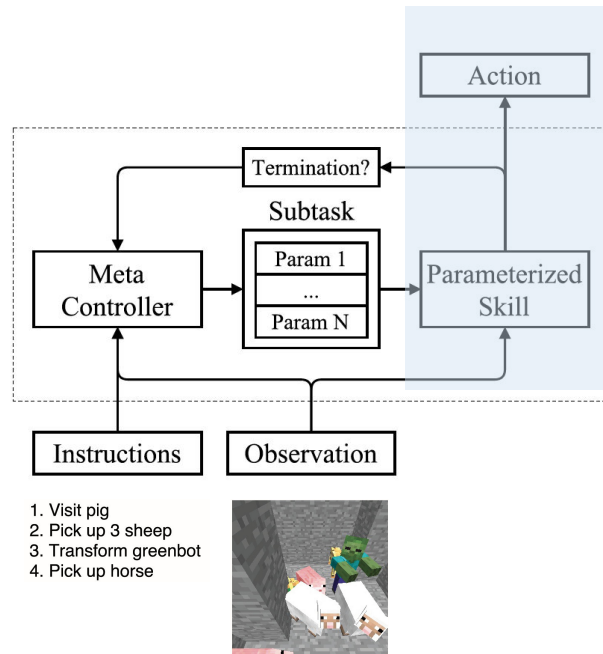


## 2) Learning to execute instructions

### Architecture

**Meta Controller:** reads instructions and passes subtask parameters to the parameterized skill.

**Parameterized skill:** execute the given subtask and

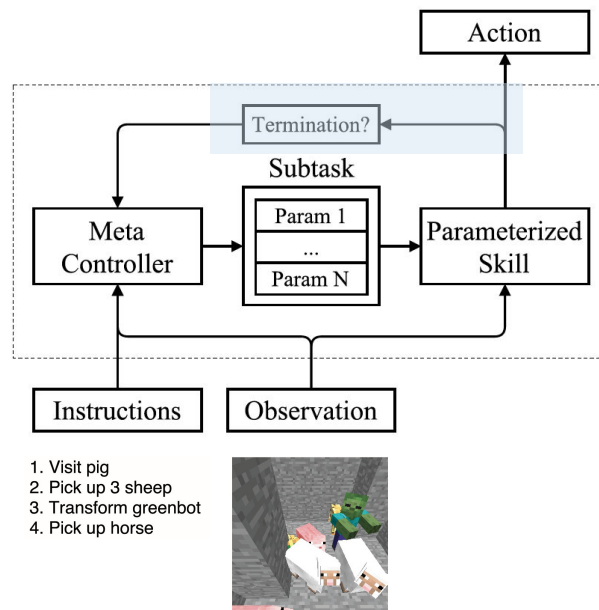


## 2) Learning to execute instructions

### Architecture

**Meta Controller:** reads instructions and passes subtask parameters to the parameterized skill.

**Parameterized skill:** execute the given subtask and gives a termination signal to the meta controller.



## 2.1) Meta Controller Architecture

Meta controller can update its subtask at any time and take the termination signal as additional input.

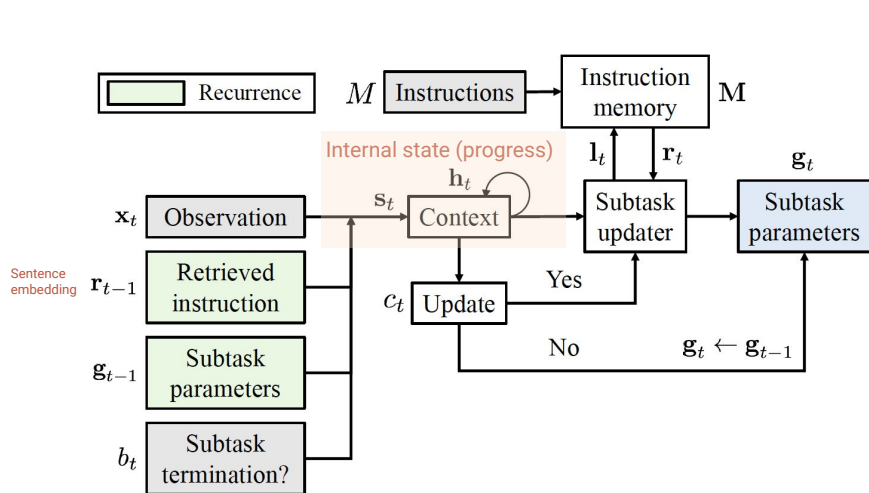
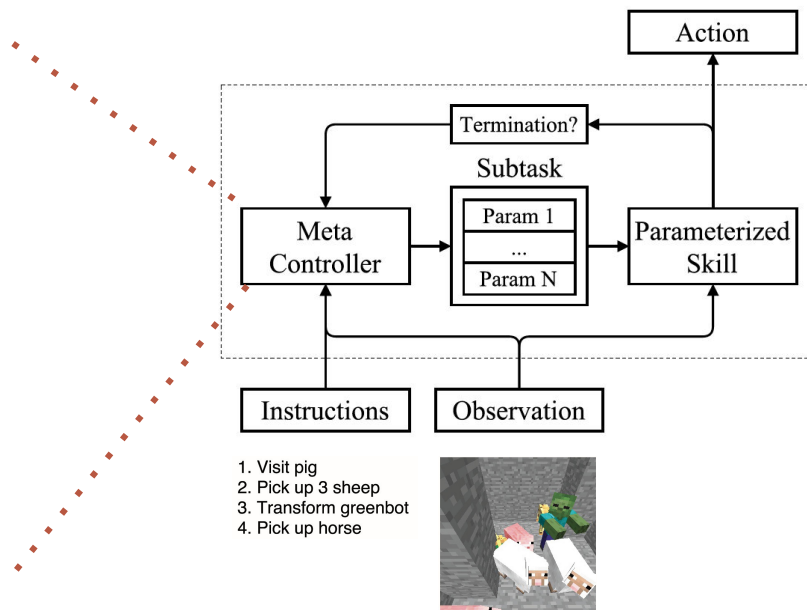


Figure 4: Neural network architecture of meta controller.



## 2.1) Meta Controller Architecture

Meta controller can update its subtask at any time and take the termination signal as additional input.

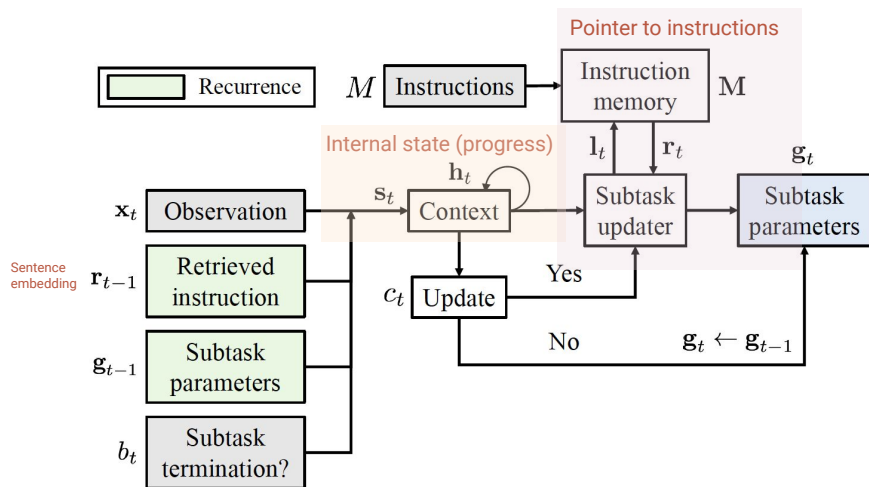
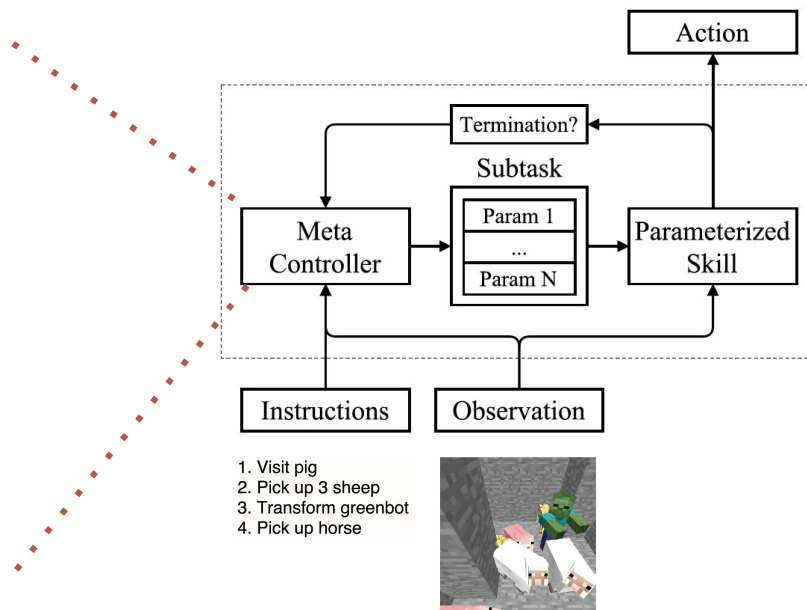


Figure 4: Neural network architecture of meta controller.

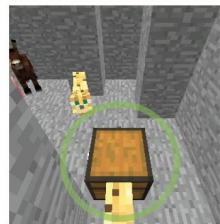


## 2) Learning to execute instructions

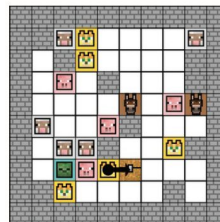
The agent needs to:

1. Execute a sequence of natural language instructions.  
Read one instruction at a time (pointer).  
Detect when the current instruction is finished.  
Memory (keep track of progress – counts)
2. **Handle unexpected events (e.g., bonus or low battery).**  
**Interrupt ongoing subtasks**

First-person-view  
(Observation)



Top-down-view  
(Not available)



### Training

1. Visit pig
2. Pick up 3 sheep
3. Transform greenbot
4. Pick up horse

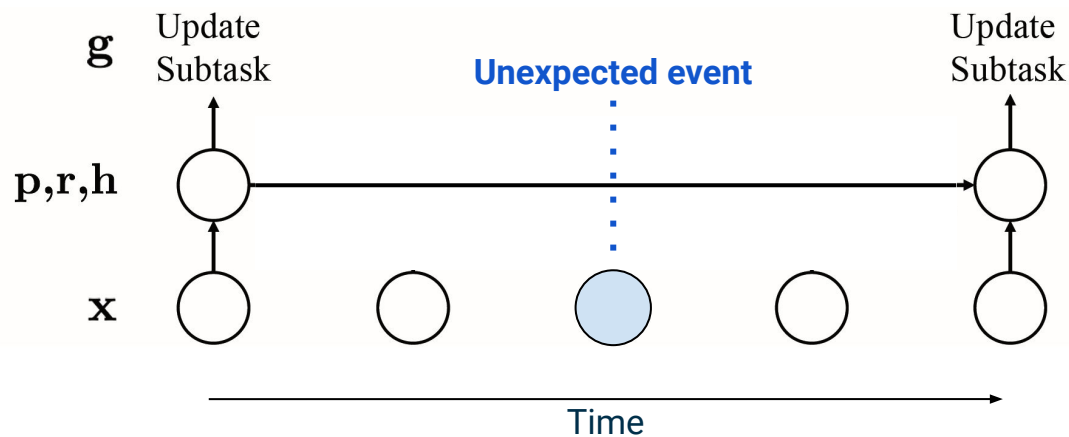
### Testing

1. Pick up pig
2. Visit sheep
3. Transform cat
4. Transform 3 sheep
5. Pick up greenbot
6. Pick up 2 pig
7. Transform 2 sheep
8. Transform 2 cat
- ...

## 2.2) Learning to Operate at a Large Time-Scale

### Open-loop meta controller

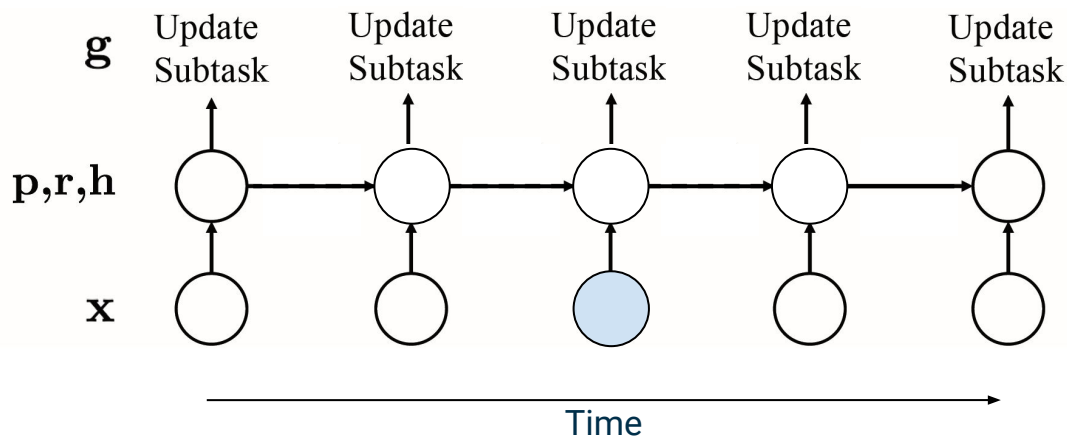
- Update subtask only when the previous one is finished.
- Pro: can operate a larger time scale.
- Con: cannot handle unexpected events immediately.



## 2.2) Learning to Operate at a Large Time-Scale

### Closed-loop meta controller

- Update subtask at every step.
- Pro: can handle unexpected events.
- Con: need to make a decision in every time step.



## 2.2) Learning to Operate at a Large Time-Scale

**Learned time-scale** for meta controller

- Meta controller **learns when to update a subtask**. It introduces an internal binary decision which indicates whether to invoke the subtask updater or not (e.g., move the pointer).

- Pro: can handle unexpected events.
- Con: can operate at larger time scale.

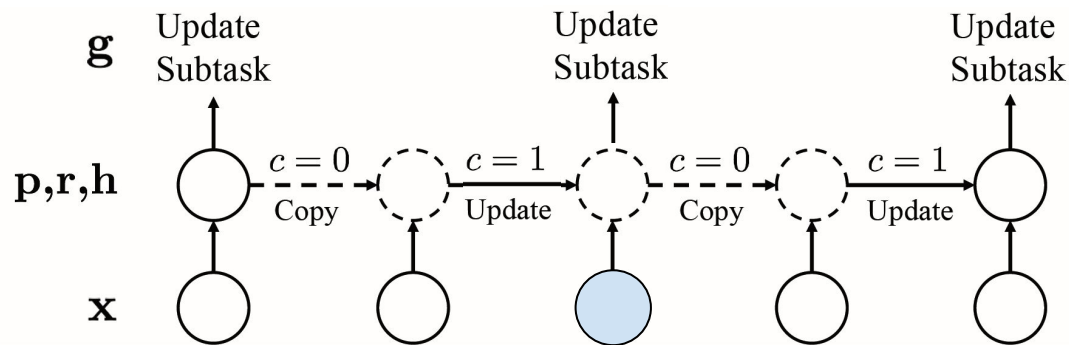


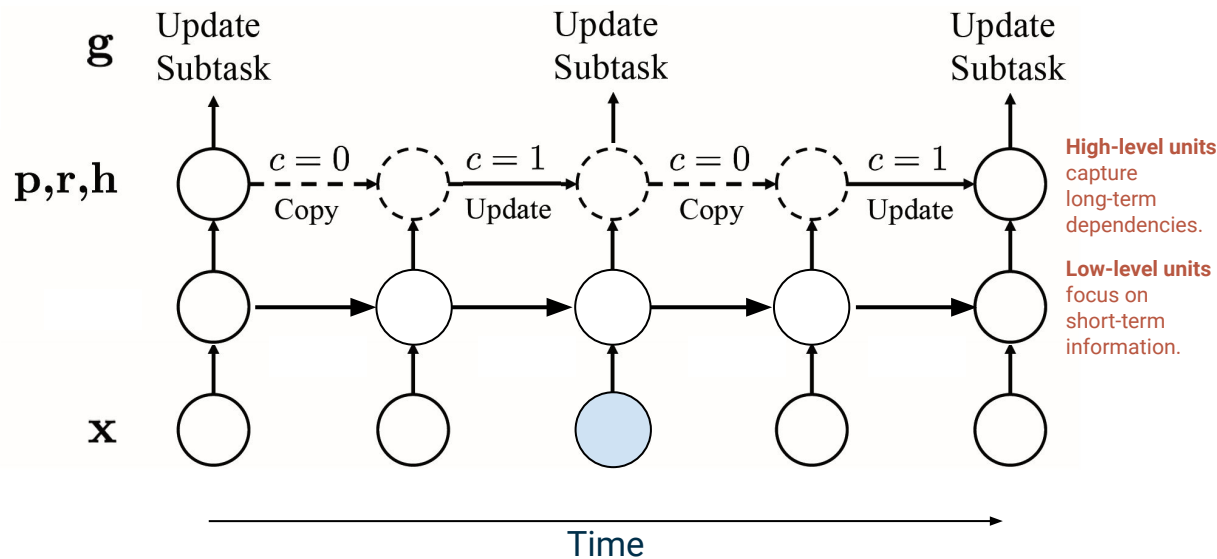
Figure 5: Unrolled illustration of the meta controller with a learned time-scale. The internal states ( $\mathbf{p}, \mathbf{r}, \mathbf{h}$ ) and the subtask ( $\mathbf{g}$ ) are updated only when  $c = 1$ . If  $c = 0$ , the meta controller continues the previous subtask without updating its internal states.



## 2.2) Learning to Operate at a Large Time-Scale

**Hierarchical dynamic time-scale** for meta controller

- Can capture both long-term and short-term temporal information.



## 2) Learning to execute instructions

### Experiments & RQs

**RQ1)** Will the proposed hierarchical architecture outperform a non-hierarchical baseline?

**RQ2)** How beneficial is the meta controller's ability to learn when to update the subtask?

## 2) Learning to execute instructions

### Experiments & RQs

Flat	It directly chooses actions <b>without using the parameterized skill</b> . It is also pre-trained on the training set of subtasks.
Hierarchical-Long	Open-loop
Hierarchical-Short	Closed-loop
<b>Hierarchical-Dynamic</b>	<b>Proposed</b> hierarchical dynamic controller.

## 2) Learning to execute instructions

### Results

	Train	Test (Seen)
Length of instructions	4	20
Without parameterized skills	Flat	-7.1 (1%) -63.6 (0%)
Open-loop	Hierarchical-Long	-5.8 (31%) -59.2 (0%)
Closed-loop	Hierarchical-Short	-3.3 (83%) -53.4 (23%)
Proposed approach	<b>Hierarchical-Dynamic</b>	<b>-3.1 (95%) -30.3 (75%)</b>

Table 2: Performance on instruction execution. Each entry shows average reward and success rate. ‘Hierarchical-Dynamic’ is our approach that learns when to update the subtask. An episode is successful only when the agent solves all instructions correctly.

## 2) Learning to execute instructions

### Results

	Train	Test (Seen)	Test (Unseen)
Length of instructions	4	20	20
Without parameterized skills	Flat	-7.1 (1%)	-63.6 (0%)
Open-loop	Hierarchical-Long	-5.8 (31%)	-59.2 (0%)
Closed-loop	Hierarchical-Short	-3.3 (83%)	-53.6 (18%)
Proposed approach	<b>Hierarchical-Dynamic</b>	<b>-30.3 (75%)</b>	<b>-38.0 (56%)</b>

Table 2: Performance on instruction execution. Each entry shows average reward and success rate. ‘Hierarchical-Dynamic’ is our approach that learns when to update the subtask. An episode is successful only when the agent solves all instructions correctly.

## 2) Learning to execute instructions

### Takeaways

- Overall performance: their agent is able to generalize to longer compositions of seen and unseen instructions by just learning to solve short sequences of a subset of instructions.
- The proposed controller is key to handle loop instructions, thanks to its ability to detect when to move to the next task (informed by parameterized skills) and keep progress in memory.
- Their architecture makes fewer decisions by operating at a large time-scale.

# Summary

Looking for: Zero-shot task generalization capabilities in Reinforcement Learning (RL)

Introduce a new RL problem with two steps:

1. An agent should learn useful skills that solve subtasks.
2. The same agent should learn to execute sequences of tasks using the learned skills.

Required generalization types:

- **Generalize to previously unseen instructions**
  - New objective which encourages learning correspondences between similar subtasks by making analogies.
- **Generalize to longer sequences of instructions**
  - Hierarchical architecture where a meta controller learns to use the acquired skills for executing the instructions.

# Takeaways

Oh et al. 2017

- Explored a type of zero-shot task generalization in RL.
  - Parameterized tasks
  - Sequence of instructions
- Propose a new problem where an agent is required to execute and generalize over sequences of instructions.
- We can teach to generalize to new tasks with analogies through metric learning (learning a distance function between objects).
- Learning when to update subtasks helps when the agent has high-level skills and deals with complex decision problems.



# Discussion prompts

1. What are the limitations of this framework? Why?
2. How does structuring losses inform learned representations?
3. How could common sense reasoning and information be injected to the model so that we don't rely as much in training analogies.
4. How do you think this architecture would generalize to other specific tasks/scenarios? Why?
5. What are some tasks that the current framework wouldn't be able to generalize? Why?