

Scalable Robot Learning in the Age of GPT

Daniel Tan

Made by Daniel Tan
<https://daniel-tan.github.io/>
Not suitable for redistribution

Generalist robots: The holy grail of robotics

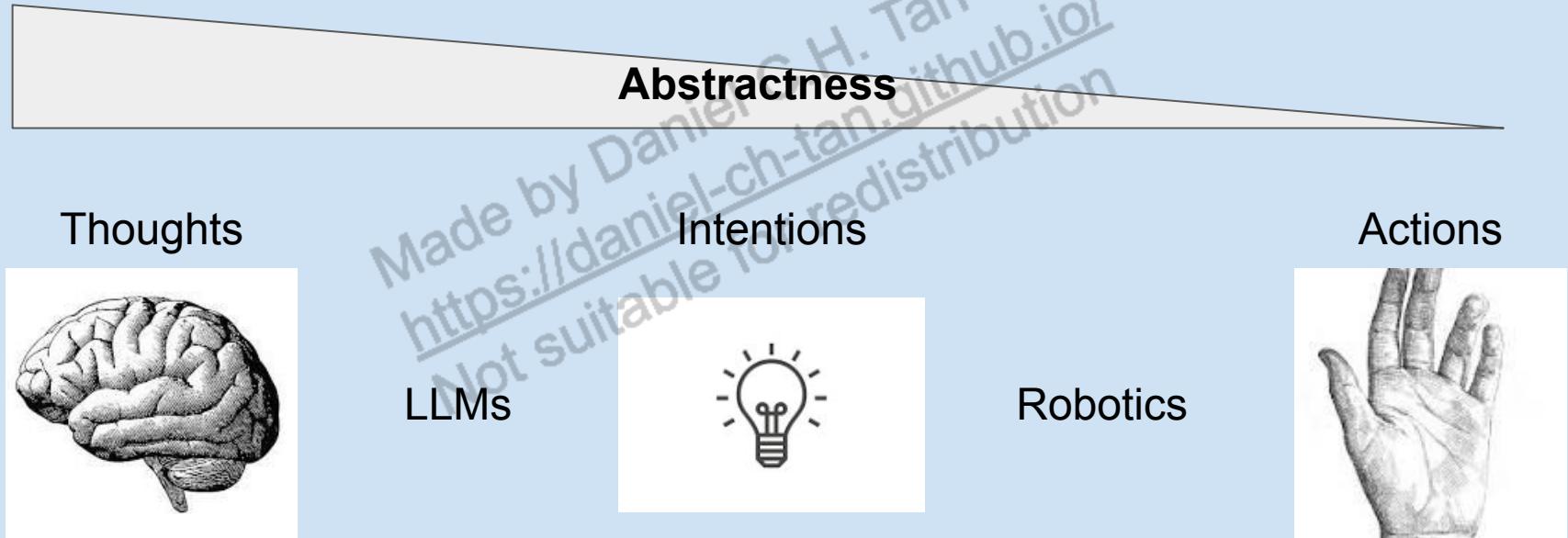
Multi-task capabilities



Diverse environments



Intelligence as a Thought-Action Pipeline



Robot learning as end-to-end visuomotor control

Robot observation



Low-level robot actions



Robot
Policy

Goal (text, image)

Multi-task imitation learning (MTIL) is surprisingly effective



Zero-shot execution of novel tasks (from NL description)



“Place grapes
in bowl”

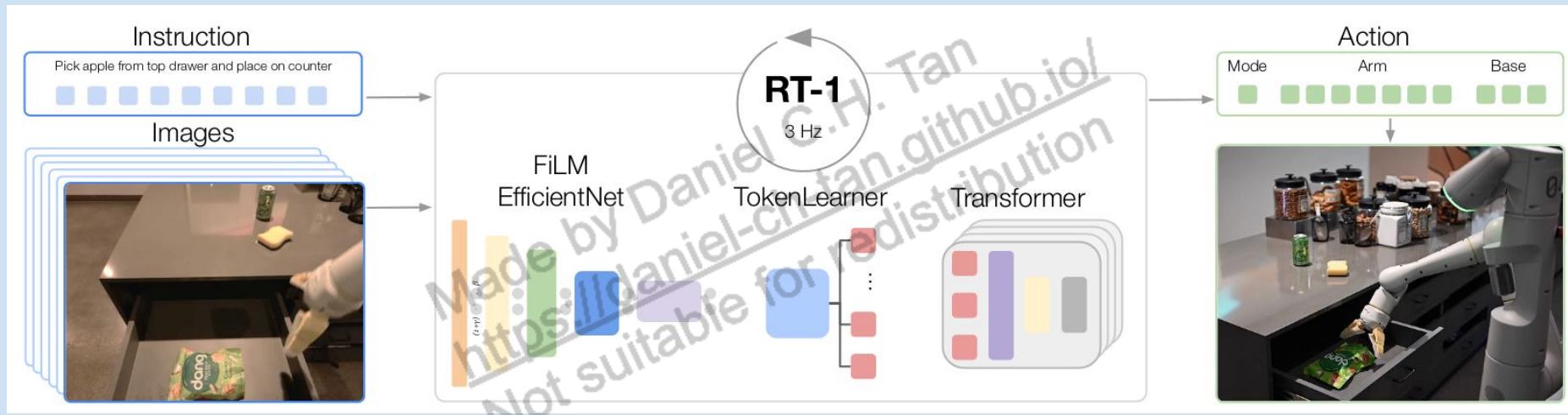


“Push bowl
across table”



“Wipe tray
with sponge”

Data + MTIL = general instruction-following models



Generalization to diverse tasks
Robust to background, distractors
Capable of long-horizon tasks

A. Brohan *et al.*, 'RT-1: Robotics Transformer for Real-World Control at Scale'. arXiv, Aug. 11, 2023. doi: [10.48550/arXiv.2212.06817](https://arxiv.org/abs/2212.06817).

Scaling up data + model size = a robot FM

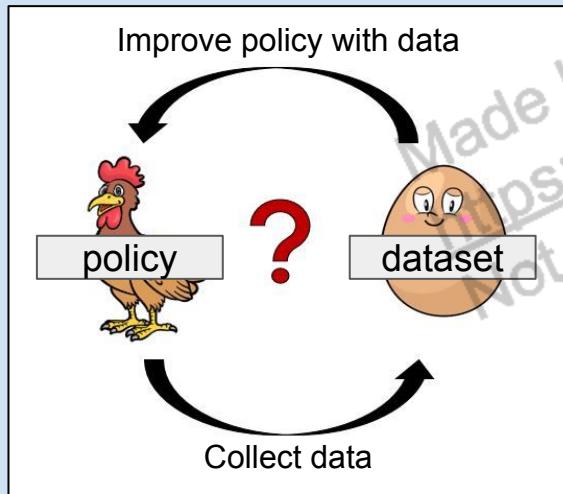


Atari games
Quadruped dogs
Robot arms

Question-answering
Image understanding

Data is the bottleneck

Need policy to get data
Need data to get policy



... data collection and curation is often the “unsung hero” of many large-scale machine learning projects...

... this is especially true in robotics, where datasets are often **robot-specific** and **gathered manually**.

A. Brohan *et al.*, ‘RT-1: Robotics Transformer for Real-World Control at Scale’. arXiv, Aug. 11, 2023. doi: [10.48550/arXiv.2212.06817](https://doi.org/10.48550/arXiv.2212.06817).

What kind of data do we need?

Semantically rich data:

Changes the environment in
interesting and meaningful ways



Open-X-Embodiment

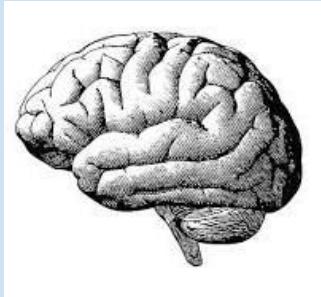
Data collected by random exploration:
Not useful for learning a good policy



RoboNet

Abstractness

Thoughts



Intentions



LLMs

- +: Text pre-training
- : Not grounded

Actions



Robotics

- +: Grounded feedback
- : Hard to scale up

- 1. How to connect LLMs to robots for decision-making?**
- 2. How to scale bottom-up robot learning?**

Remainder of talk

1. **How can LLMs accelerate robot learning?**
2. An end-to-end example in MiniGrid
3. Future work

Made by Daniel C.H. Tan
<https://daniel-ch-tan.github.io/>
Not suitable for redistribution

A note on LLM perception...

Robot observation:
Low-level sensor data



*Visual, tactile,
proprioceptive, ...*

LLM input:
Text

You are hungry! Let's cook a delicious meal. Check the cookbook in the kitchen for the recipe. Once done, enjoy your meal!

-= Backyard =-
You find yourself in a backyard.

You make out a patio table. But the thing is empty. You see a patio chair. Wow, isn't TextWorld just the best? The patio chair is stylish. But there isn't a thing on it. You see a gleam over in a corner, where you can see a BBQ.

There is a closed screen door leading south. There is a closed wooden door leading west. There is an exit to the east. Don't worry, there is no door.

Perception is solvable (eventually)

Scalability

Manual (text)
description pipeline

E.g. NetHack Language
Wrapper

Observation:

- vertical wall far east
- stairs up far westsouthwest
- horizontal wall near north and northeast
- lichen near northnortheast
- fountain near eastnortheast
- dark area near southwest
- gold piece very near northnortheast
- horizontal wall very near southeast and south
- doorway adjacent southwest
- vertical wall adjacent west and northwest

Visual programming:
Write code to parse scene

Can use a VLM, e.g. CLIP

Multimodal
LLMs

IMAGE: 

Question: Are there both ties and glasses in the picture?

Program:

```
BOX0=Loc(image=IMAGE, object='ties')
ANSWER0=Count(box=BOX0)
BOX1=Loc(image=IMAGE, object='glasses')
ANSWER1=Count(box=BOX1)
ANSWER2=Eval("yes" if {ANSWER0} > 0 and {ANSWER1} > 0 else "no")
RESULT=ANSWER2
```

Prediction: no



GPT - 4

What is funny about this image? Describe it panel by panel.



GPT-4 understands visual humour!

The image shows a package for a "Lightning Cable" adapter with three panels.

Panel 1: A smartphone with a VGA connector (a large, blue, 15-pin connector typically used for computer monitors) plugged into its charging port.

Panel 2: The package for the "Lightning Cable" adapter with a picture of a VGA connector on it.

Panel 3: A close-up of the VGA connector with a small Lightning connector (used for charging iPhones and other Apple devices) at the end.

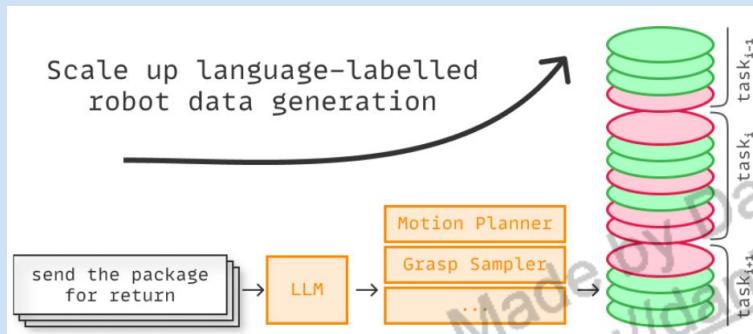
The humor in this image comes from the absurdity of plugging a large, outdated VGA connector into a small, modern smartphone charging port.

→ Future LLMs will understand sensor data *without* perception pipelines

→ “Hacky” methods now are okay as a proxy for future capabilities

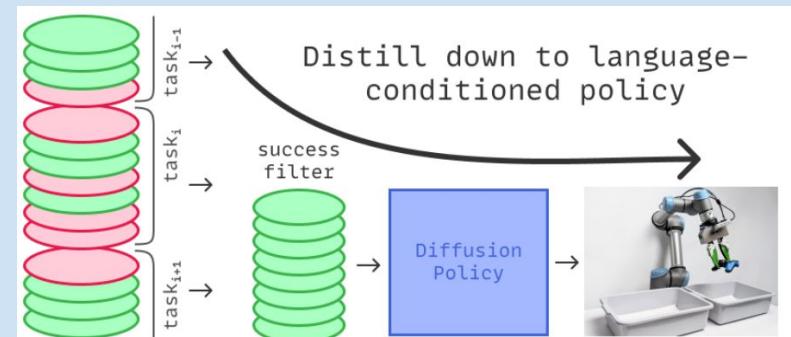
OpenAI, ‘GPT-4 Technical Report’. arXiv, Mar. 27, 2023. doi: [10.48550/arXiv.2303.08774](https://doi.org/10.48550/arXiv.2303.08774).

We don't need perfection from LLMs

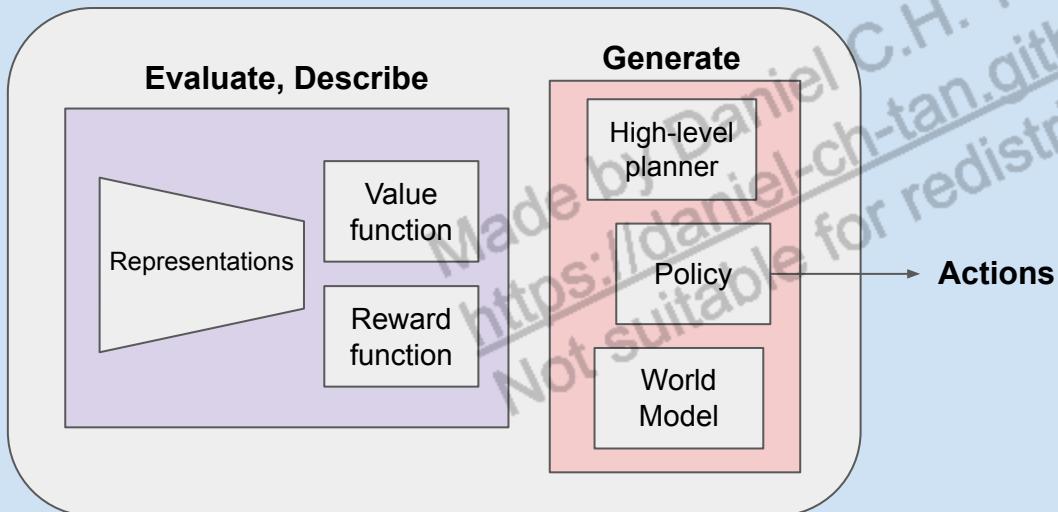


Filter the data for success.
Distill that into a performant policy!
(*we need some success)

Treat LLM-based controllers as a stream
of diverse, *semantically rich* data.
(some good, some bad)



How do LLMs fit into the robot learning pipeline?



We will consider complex, LLM-powered robot control systems.

The LLM may only be a component; the whole system defines a policy.

High-level Planner

Given the current state, choose an *abstract* action.

Robot observation



Planner

Abstract action

“Pick up the
bowl”

Why is this valuable?

→ Decompose the learning problem to simplify the low-level policy

→ High-level plans are embodiment-agnostic: can be inferred from human demonstrations; reused for different robots

Case study: MineDojo

Sim. environment with 1000s of language-prompted tasks in MineCraft



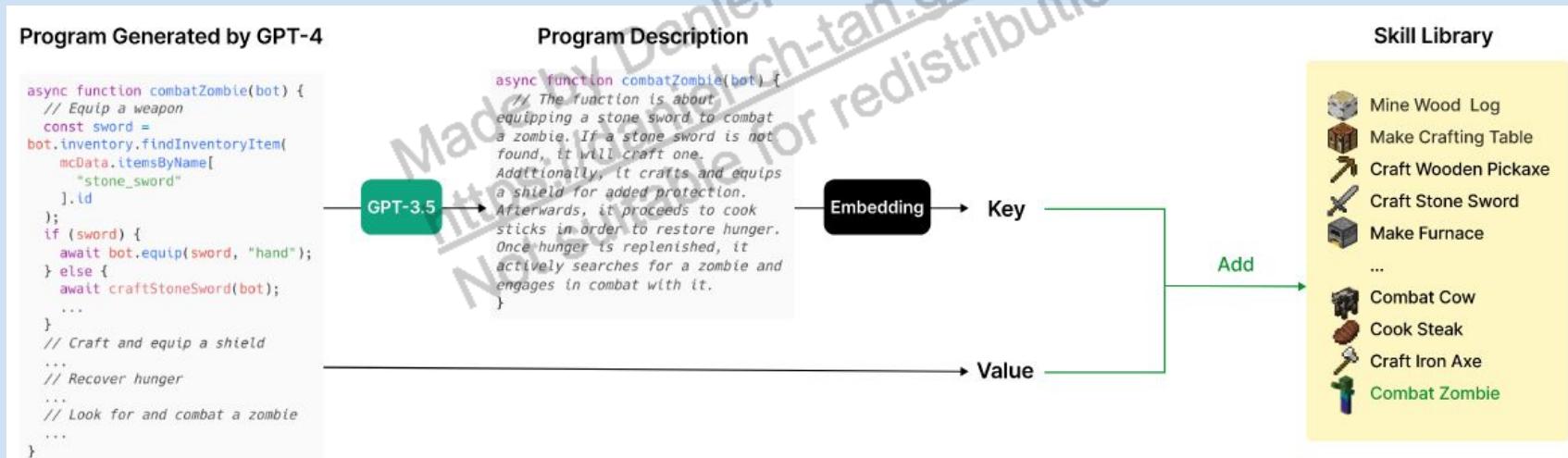
Some are very difficult!

- Fight an Ender Dragon
- Build and enter a Nether Portal



Skills defined as code

- Easy for LLM to read, understand
- Can be easily re-used in subsequent skills



Proposing “interesting” goals

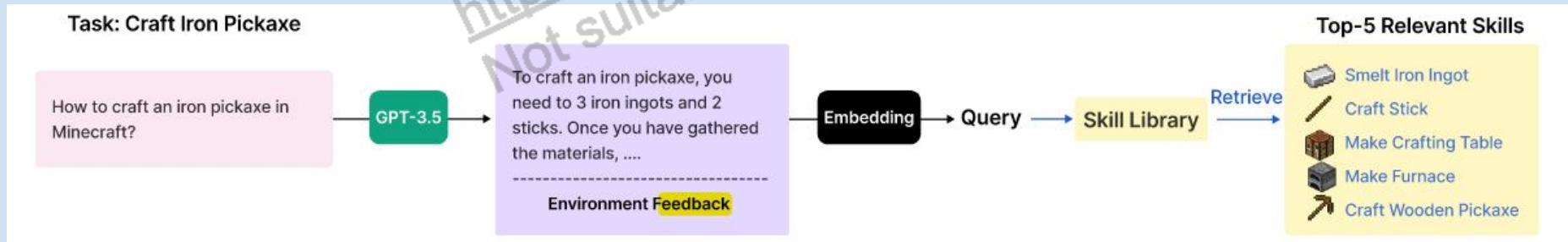
High-level goal: “Discover as many diverse things as possible”
→ leverage LLM’s semanticity and judgement!

| | | | |
|---|---|--------------|---|
|  | <p>Inventory (5/36): {oak_planks: 3, 'stick': 4, 'crafting_table': 1, 'stone': 3, 'wooden_pickaxe': 1}</p> | <p>GPT-4</p> | <p>Reasoning: Since you have a wooden pickaxe and some stones, it would be beneficial to upgrade your pickaxe to a stone pickaxe for better efficiency. Task: Craft 1 stone pickaxe.</p> |
|  | <p>Biome: river Inventory (4/36): {'sugar_cane': 3, 'paper': 3, 'fishing_rod': 1, 'wooden_pickaxe': 1}</p> | <p>GPT-4</p> | <p>Reasoning: Since you have a fishing rod in your inventory and you are near a river biome, it would be a good opportunity to catch some fish for food and experience. Task: Catch 1 fish.</p> |
|  | <p>Nearby entities: pig, cat, villager Health: 12/20 Hunger: 0/20</p> | <p>GPT-4</p> | <p>Reasoning: Your hunger is at 0, which means you need to find food to replenish your hunger. Since there are pigs nearby, you can kill one to obtain raw porkchops. Task: Kill 1 pig.</p> |
|  | <p>Inventory (6/36): {'furnace': 1, 'stone_pickaxe': 1, 'oak_planks': 7, 'crafting_table': 1, 'raw_iron': 4, 'coal': 1}</p> | <p>GPT-4</p> | <p>Reasoning: You have raw iron and coal, and you have a furnace. It's time to smelt the iron to make iron ingots, which can be used to craft better tools and armor. Task: Smelt 4 raw iron.</p> |
|  | <p>Time: night Nearby entities: zombie Equipment: 'stone_sword', 'shield'</p> | <p>GPT-4</p> | <p>Reasoning: Since it's night and there's a zombie nearby, it's a good opportunity to try killing the zombie now that you have a stone sword and a shield equipped. Task: Kill 1 zombie.</p> |

Self-QA for reasoning, grounding

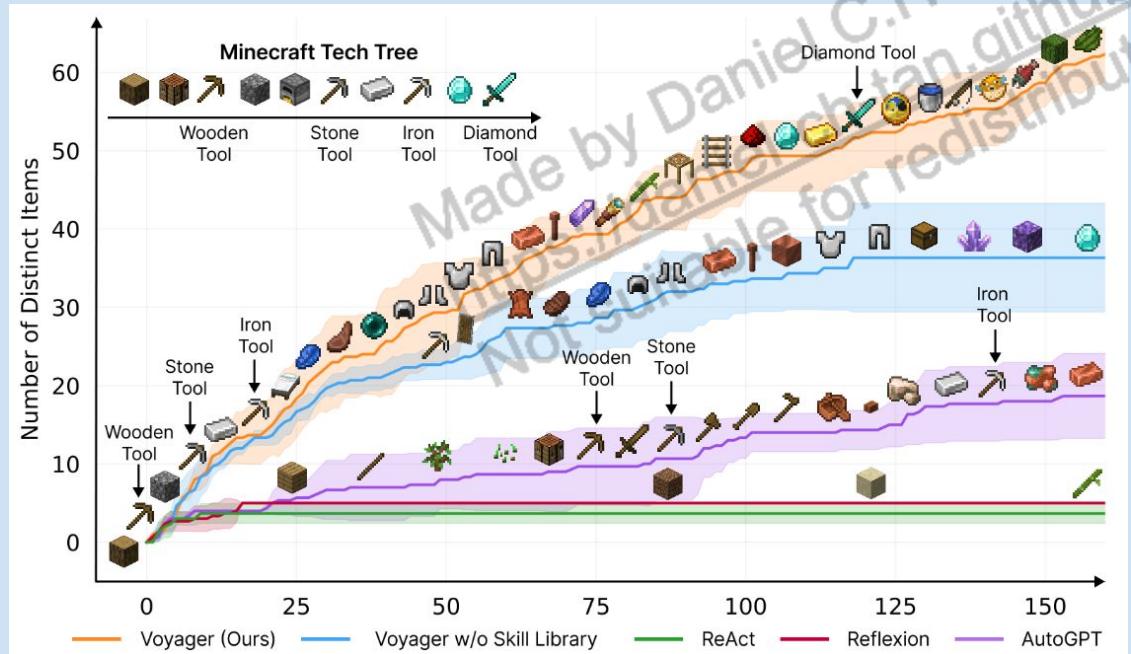
LLM can self-ask questions based on agent state
→ elicit reasoning
→ C.f. chain of thought

LLM can self-answer questions with wiki lookup
→ incorporate prior knowledge!



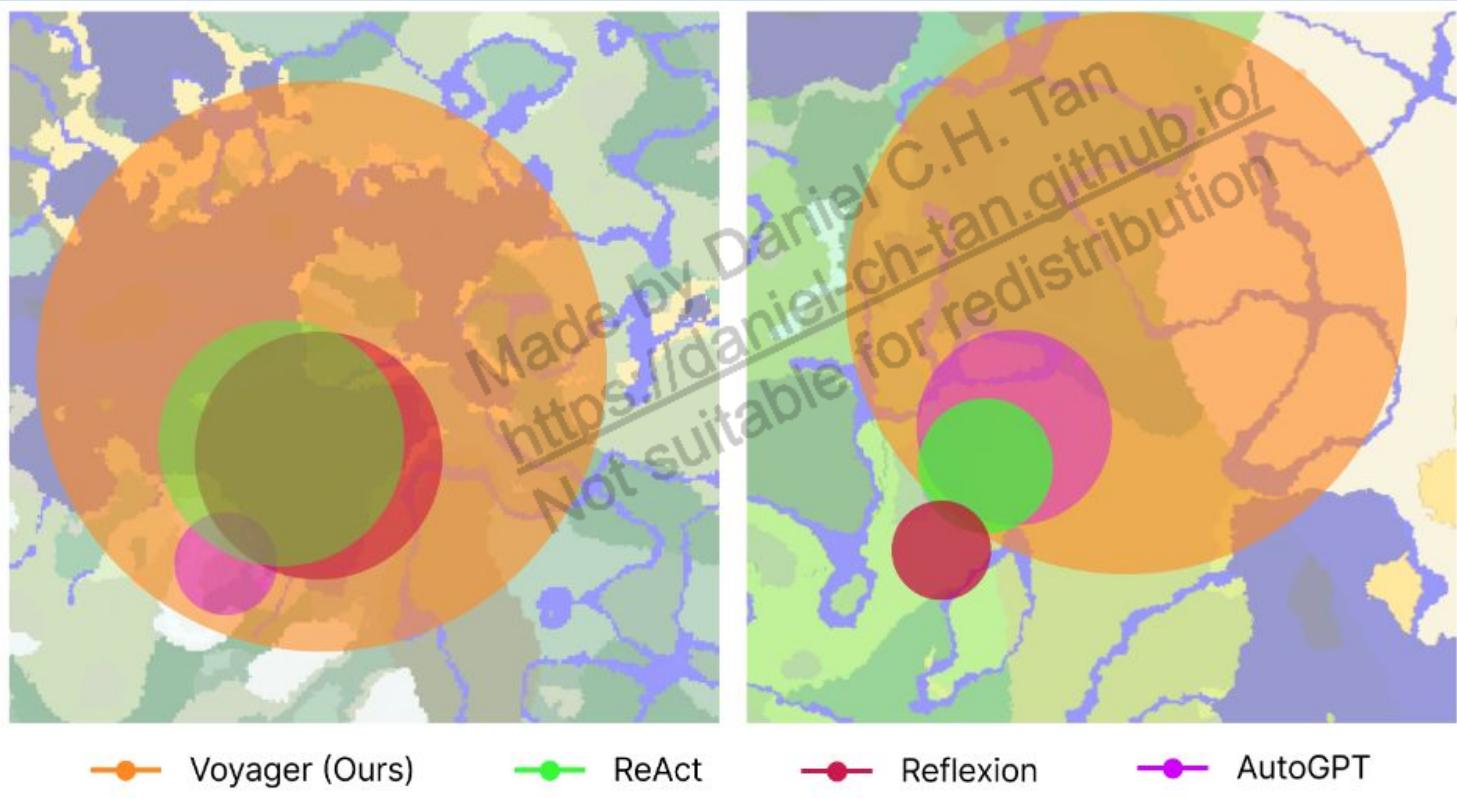
Open-ended learning of complex behaviours

Self-driven discovery and achievement
of increasingly complex goals



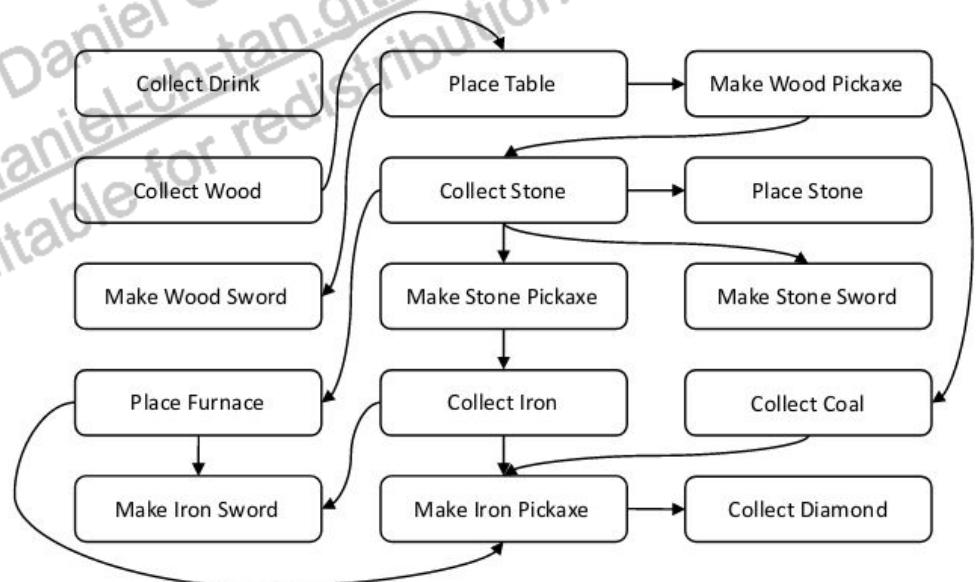
Among other things,
LLM learns to:
→ Forge armor, shields
→ Mine diamonds
→ Fish for puffers

Visiting diverse world states



Defining “interesting” curricula for RL

You are a player in a game. You want to learn as many different skills as possible. You can do these tasks well: <tasks done well>. Suggest whether the given tasks are interesting: <tasks to be determined>.

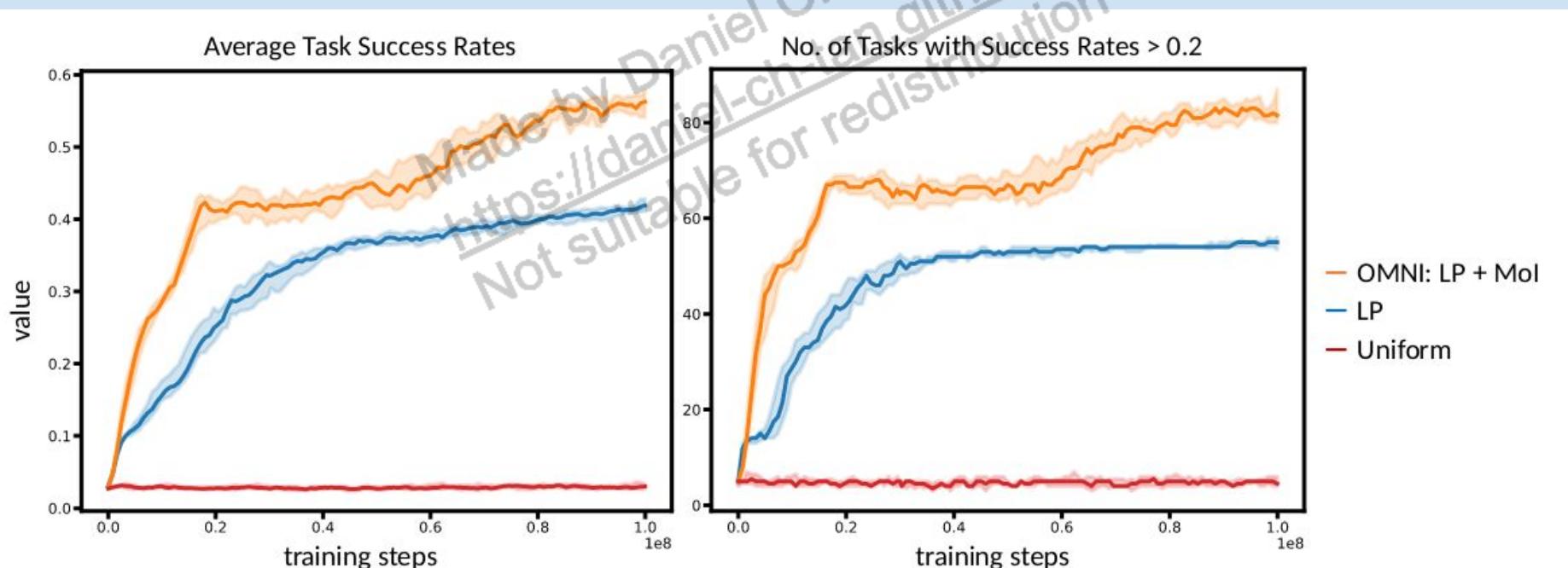


“Interestingness” outperforms non-semantic methods

LLMs can identify tasks that expand agent’s skill set

Non-semantic methods do not have task-skill mapping

→ “waste time” on redundant tasks

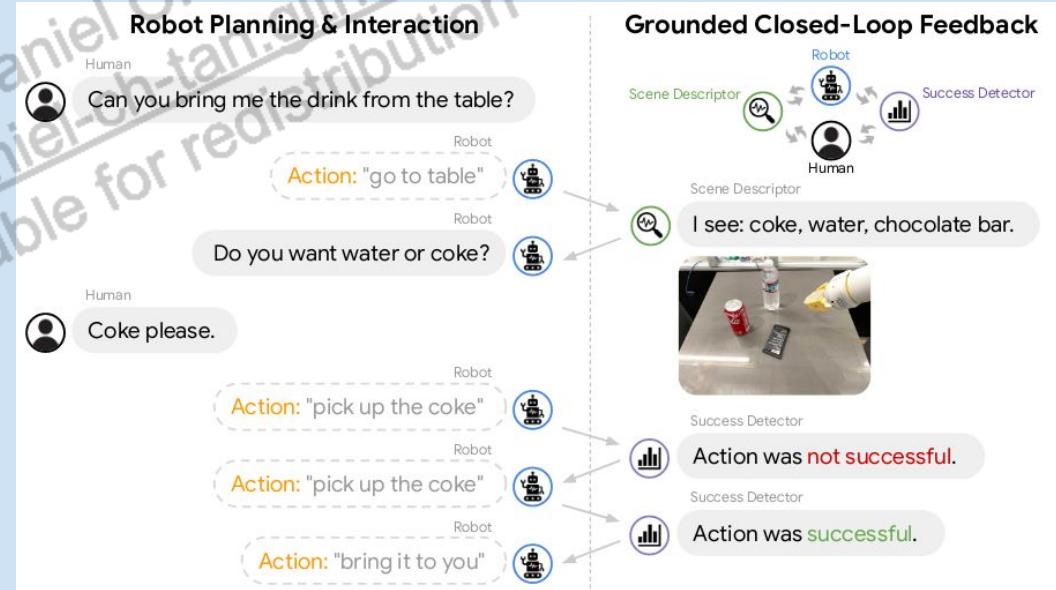


LLM planning over pre-defined skills

SayCan: Open-loop planning
over pre-defined “skills”

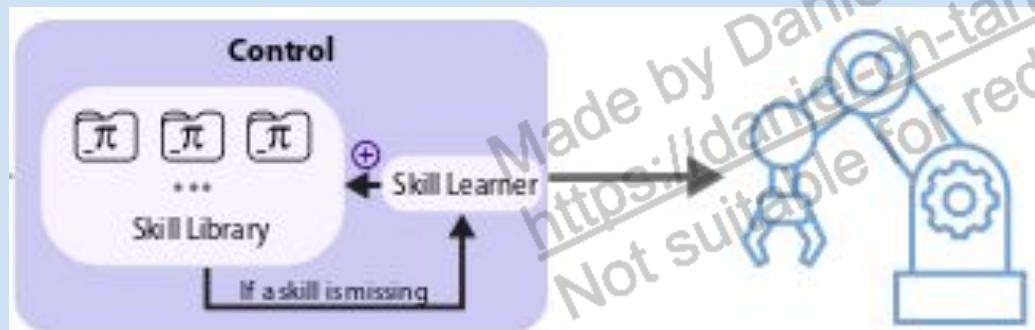


Inner Monologue: Closed-loop planning.
Adapts to unplanned situations!



What about missing skills?

HalP: Simply allow LLM to ask for a new skill!



“Learn_skill(skill)” is a tool the LLM can use.

→ This prompts a human to collect demonstrations
→ Efficiently learn a policy (Neural Descriptor Field) for the skill

→ Negative reward when using the skill incorrectly

Infeasible plans

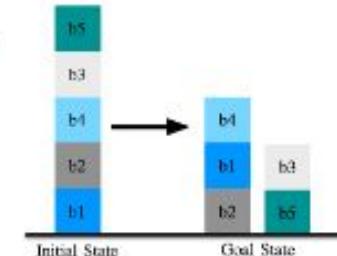
Problem (P1): You have 5 blocks. One cannot place more than one block on another block. b5 is on top of b3. b4 is on top of b2. b2 is on top of b1. b3 is on top of b4. b1 is on the table. b5 is clear. Your arm is empty.

Your goal is to move the blocks.

b1 should be on top of b2.

b3 should be on top of b5.

b4 should be on top of b1.



GPT-4:

1. Move b5 from b3 to the table.
2. Move b3 from b4 to b5.
3. Move b4 from b2 to the table.
4. Move b2 from b1 to b5.
5. Move b1 from the table to b2.
6. Move b4 from the table to b1.

Here we've put B3 on B5.

So, in a later step,
we can't put B2 on B5!

Use a planner instead!

LLM+P: “Outsource” reasoning to planning solver

An Example PDDL Problem File Written by GPT-4

Prompt: Description of Problem (P1) + Provide me with the problem PDDL file that describes the planning problem directly without further explanations.

GPT-4 (the generated problem PDDL):

```
(:objects b1 b2 b3 b4 b5 - block)
(:init (on b5 b3) (on b4 b2) (on b2
b1) (on b3 b4) (clear b5) (empty))
(:goal (and (on b1 b2) (on b3 b5)
(on b4 b1)))
```

LLM+P outperforms all pure LLM approaches

| Domain | Success Rate % | | | | |
|-------------|------------------|---------|--------------------|--------------------|-----------------|
| | LLM ⁻ | LLM | LLM ^{ToT} | LLM+P ⁻ | LLM+P |
| BARMAN | 0 | 0 | 0 | 0 | 20 (100) |
| BLOCKSWORLD | 20 | 15 (30) | 0 (5) | 0 | 90 |
| FLOORTILE | 0 | 0 | 0 | 0 | 0 |
| GRIPPERS | 25 (60) | 35 (50) | 10 (20) | 0 | 95 (100) |
| STORAGE | 0 | 0 (25) | 0 | 0 | 85 |
| TERMES | 0 | 0 | 0 | 0 | 20 |
| TYREWORLD | 5 | 15 | 0 | 0 | 10 (90) |

TABLE I: Success rate % of applying LLM-AS-P without context (LLM⁻), LLM-AS-P (LLM), Tree of Thoughts (LLM^{ToT}), LLM+P without context (LLM⁻), and LLM+P.

- Translation is easier than reasoning
- Benefits from guarantees of PDDL
- Implementation-wise: easier to check correctness (of translation vs planning)

Takeaways here

- LLMs propose semantic *goals* or *curricula* → effective *exploration*
- LLMs can plan over tools
 - Code descriptions
 - Language descriptions
- “Learn a skill” is also a tool!

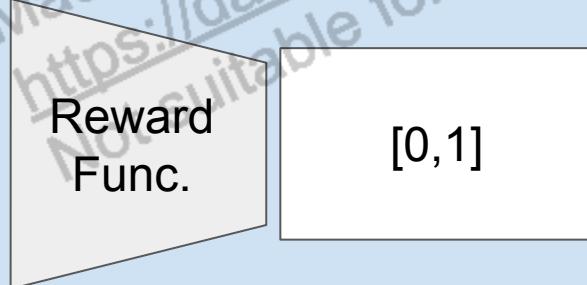
Hallucination remains an issue when asking an LLM to directly output plans

- Workaround: Output a specification, solve with PDDL

Reward Function

Given the current state, assign a *reward*

Robot observation



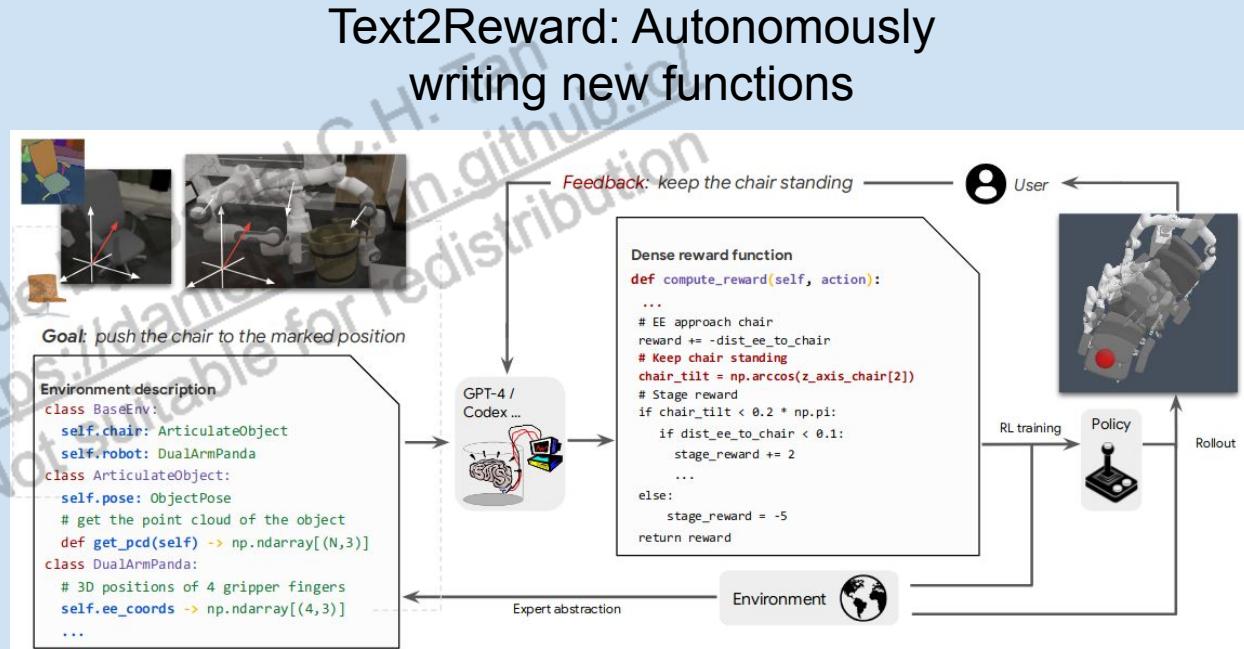
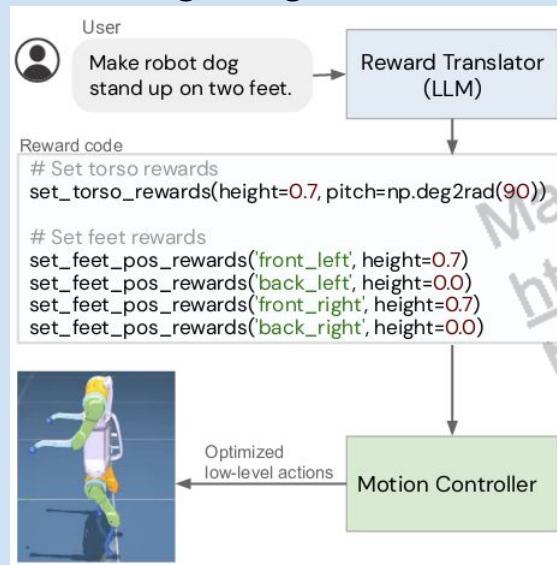
Why is this valuable?

- Evaluation is easier than generation (of a plan)
- Grounding is less of an issue, since we already train on and evaluate for QA
- Rewards are also embodiment-agnostic

Goal-oriented rewards can be obtained from planner!

Coding a goal-oriented reward function

Language2Reward: Configuring functions



Quadruped learns to stand on hind legs

Generated reward outperforms oracle reward in MetaWorld

QA for semantic intrinsic reward (IR)

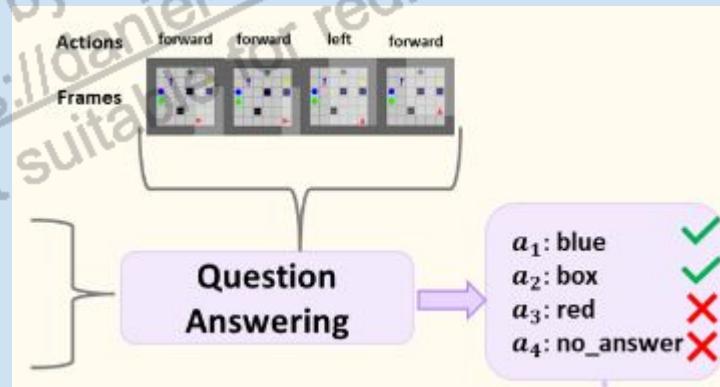
Define task-relevant questions (e.g. with LLM)

Goal: put the blue box next to the grey ball

Question Generation

- q_1 : put the __ box next to the grey ball
- q_2 : put the blue __ next to the grey ball
- q_3 : put the blue box next to the __ ball
- q_4 : put the blue box next to the grey __

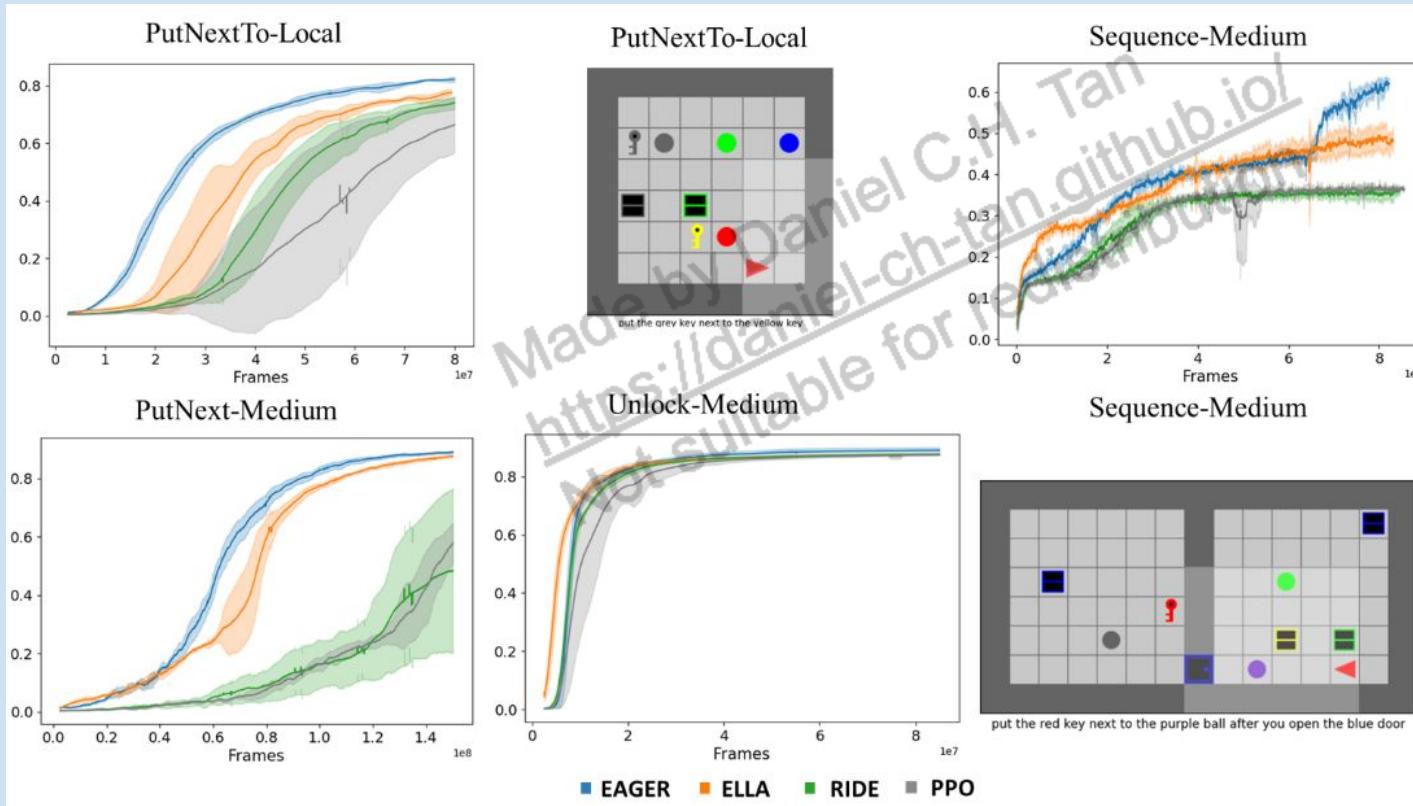
Try to answer questions given agent's trajectory (e.g. with LLM)
→ reward if correct



Incentivizes agent to explore for information!

Note: LLM not used in this paper

Semantic IR outperforms non-semantic IRs



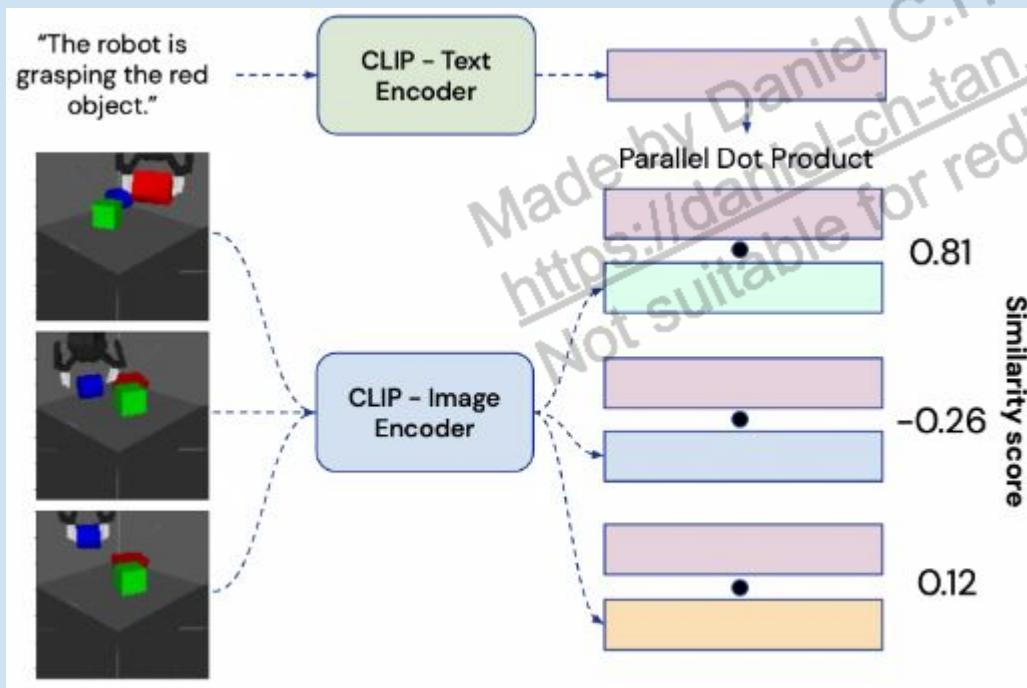
EAGER:
Semantic IR
derived from QA

RIDE, ELLA:
Non-semantic
baseline IRs

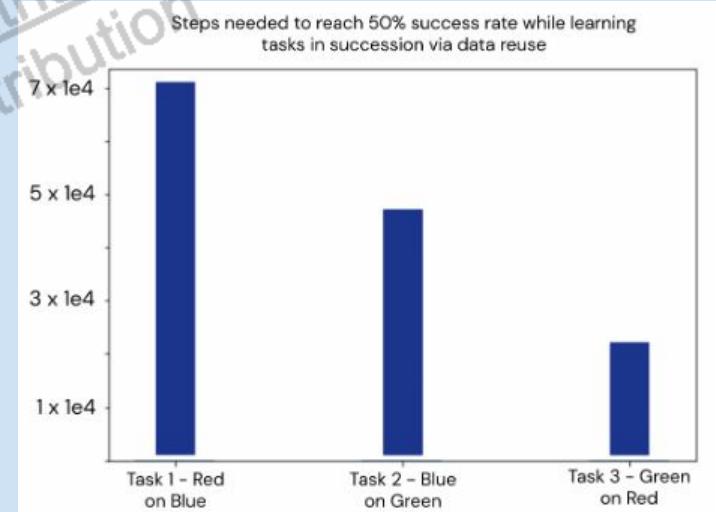
Note: hybrid IRs
likely better than
pure IRs

Re-annotating prior data

An LLM could be used to re-annotate existing data with achieved goals



Re-using data improves sample-efficiency of learning RL policy (from scratch)



Takeaways here

- LLMs can robustly translate NL to code reward functions
- LLMs can propose semantic *intrinsic rewards*
- LLMs can repurpose existing data for new tasks

Made by Daniel Ch-Tan
<https://daniel-ch-tan.github.io/>
Not suitable for redistribution

Zero-shot vs Fine-tuned

So far: High-level planners, reward functions

- Can be obtained zero-shot (from internet pre-training)
- Abstracted beyond low-level robot data
- Can use APIs since only inferences needed

Upcoming: World models, low-level policies

- Likely require fine-tuning.
- Directly involve low-level robot information
- Need parameter access to model → trickier to set up

World Models

Given the current state, predict the future states

Robot observation



World Model

Robot action



Future observation

Why is this valuable?

→ Language: rich prior knowledge about dynamics which we should try to exploit!

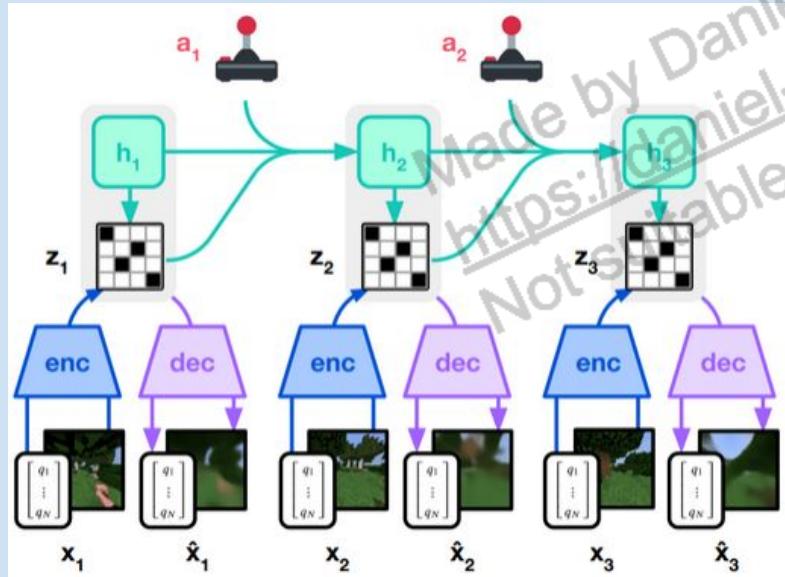
→ Dynamics are task agnostic. A single WM applies to 1000s of tasks

Preface: World Models

Encode obs to latent state

Predict future latent state given action

Decode future latent to future obs



The same architecture is
SoTA in diverse domains



DreamerV3

What if we just added
language?

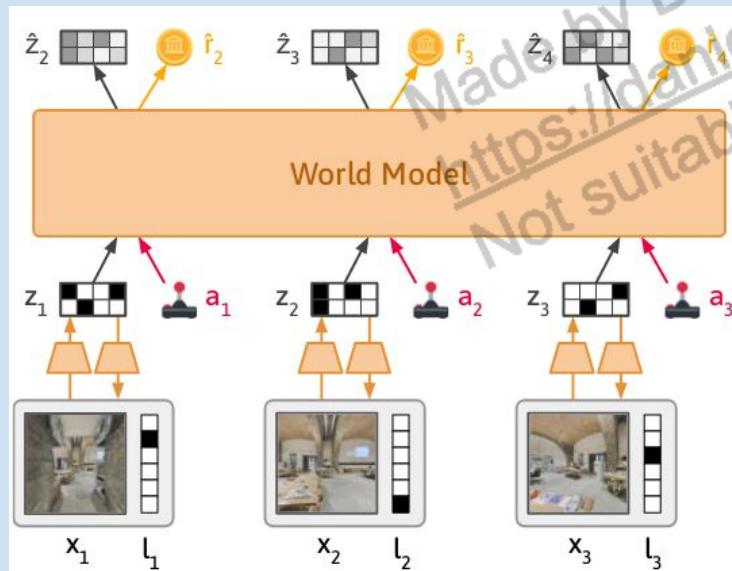
DynaLang: Fine-tuning a WM with LLM objective

Encode obs + lang to latent state

Predict future latent given action

Decode future latent to next obs +
next lang

A joint architecture for video
prediction and language
modelling



- Can be pre-trained on either modality alone!
- Signs of positive transfer between modalities

Using language to predict the future...

the **bottle** is in the living room

Info about the world
→ helps predict future states

get the **bottle**

Info about the task
→ helps predict future rewards

Video prediction



Reward prediction

$r=0$

$r=0$

$r=0$

$r=0$

$r=1$

Using language to predict the future...

the **bottle** is in the living room

Info about the world
→ helps predict future states

get the **bottle**

Info about the task
→ helps predict future rewards

Video prediction



Reward prediction

$r=0$

$r=0$

$r=0$

$r=0$

$r=1$

...and using the past to predict language

Video and text inputs



Text prediction

kitchen

the **plates** are in the

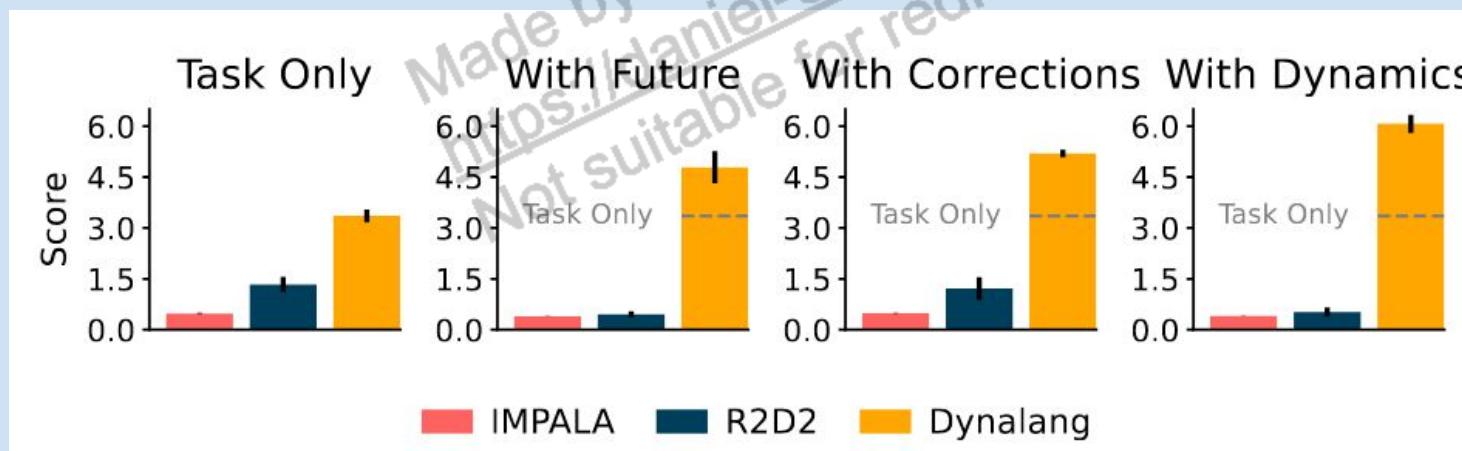
DynaLang benefits from diverse language hints

“Put the bottle in the trash bin”

“The bottle is in
the living room”

“Go the other
way”

“If you step on
the pedal, the bin
opens”



Can an LLM-WM benefit from prior (text) knowledge?



GAME 1 MANUAL

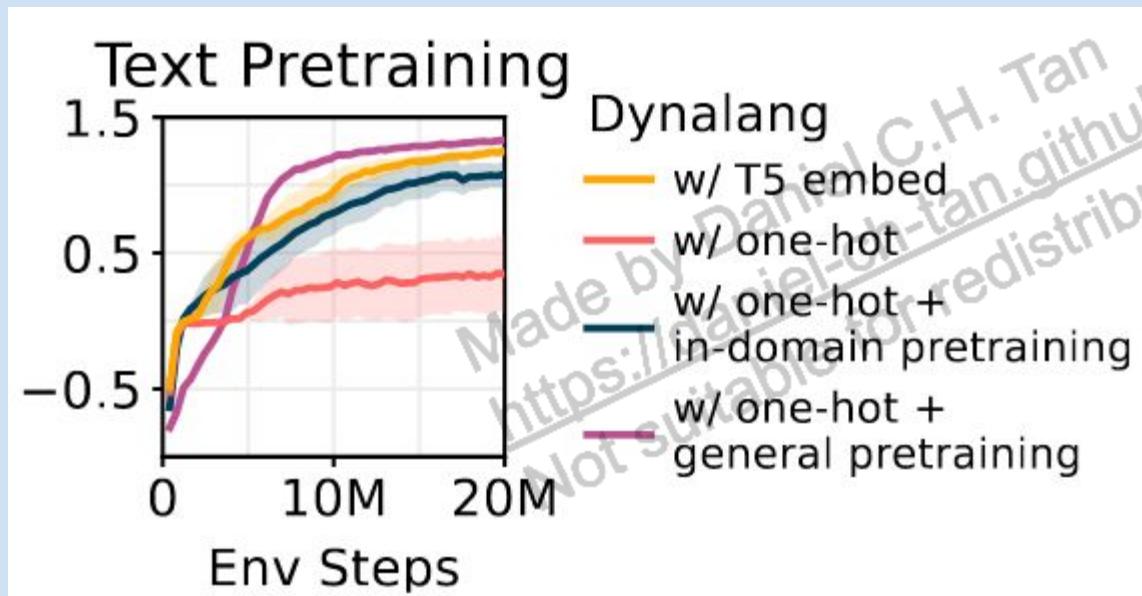
1. at a particular locale, there exists a motionless mongrel that is a formidable adversary.
2. the top-secret paperwork is in the crook's possession, and he's heading closer and closer to where you are.
3. the crucial target is held by the wizard and the wizard is fleeing from you.
4. the mugger rushing away is the opposition posing a serious threat.
5. the thing that is not able to move is the mage who possesses the enemy that is deadly.
6. *the vital goal is found with the canine, but it is running away from you.*

Messenger:
2D grid-based game

Each instance has a *game manual* describing the rules and goals

Pre-training of LLM-WM:
Mask out the visual
encoder-decoder activations

Model benefits from general pre-training



In-domain text: Game manuals for Messenger

Out-of-domain text: 2M short stories generated by GPT

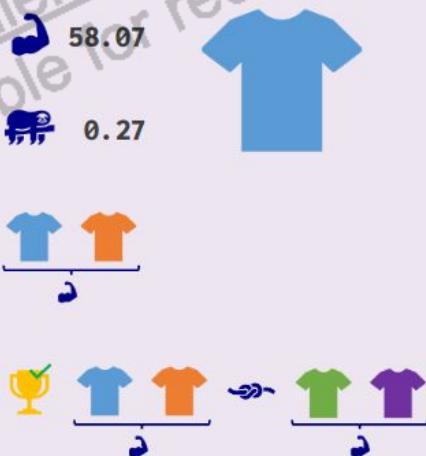
General pre-training outperforms domain-specific pre-training!
→ Signs of life for scaling up

Translating to probabilistic programs

Define atoms and relations for a *generative world model* (e.g. prob-prog)

(A) Generative world model

```
(define strength (mem (lambda (player)
  (gaussian 50 20))))  
  
(define laziness (mem (lambda (player)
  (uniform 0 1))))  
  
(define (team-strength team)
  (sum (map (lambda (player)
    (if (flip (laziness player))
        (/ (strength player) 2)
        (strength player)))
    team)))  
  
(define (won-against team-1 team-2)
  (> (team-strength team-1) (team-strength team-2)))
```



Consider: A tug of war game between teams

Propose two attributes: player **strength** and **laziness**

Define a generative process for determining which team wins, given attributes

Translating to probabilistic programs

LLMs can then translate NL queries into prob-prog queries to do *inference*

Josh won against Lio.

Josh proceeded to claim victory against Alex.

Even working as a team, Lio and Alex still could not beat Josh.

How strong is Josh?

Not suitable for redistribution
<https://daniel-ch-tan.github.io/>

(condition (won-against '(josh) '(lio)))
(condition (won-against '(josh) '(alex)))
(condition (not (won-against '(lio alex)
 '(josh))))

(query (strength 'josh))



Prior: $\mu = 50.00\%$
Josh: $\mu = 67.83\%$

Similar idea of “outsourcing” predictions to a specialised module

→ Note: predict abstract attributes (vs low-level state)

Takeaways here

- LLMs can leverage diverse textual information for predicting future states
- LLMs can identify semantically-relevant *atoms* and propose common-sense *relations* to define a *generative model*
 - Exist methods for fast approximate inference!

Using NL information to inform RL is very exciting.

More work needed in this direction!

Policies

Given the current state, predict a *robot action*

Robot observation



Reward
Func.

Robot action



Why is this valuable?

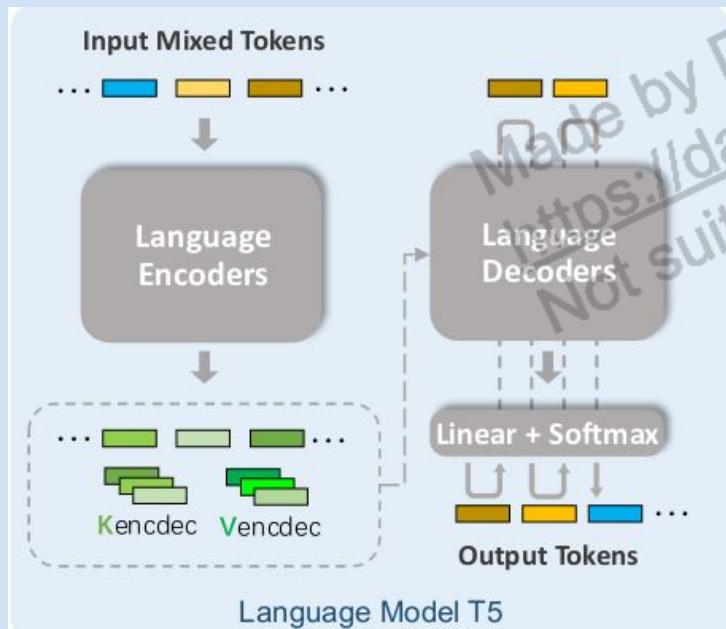
→ This is directly what we want!

Previous approaches:
policy through subsequent
IL, RL

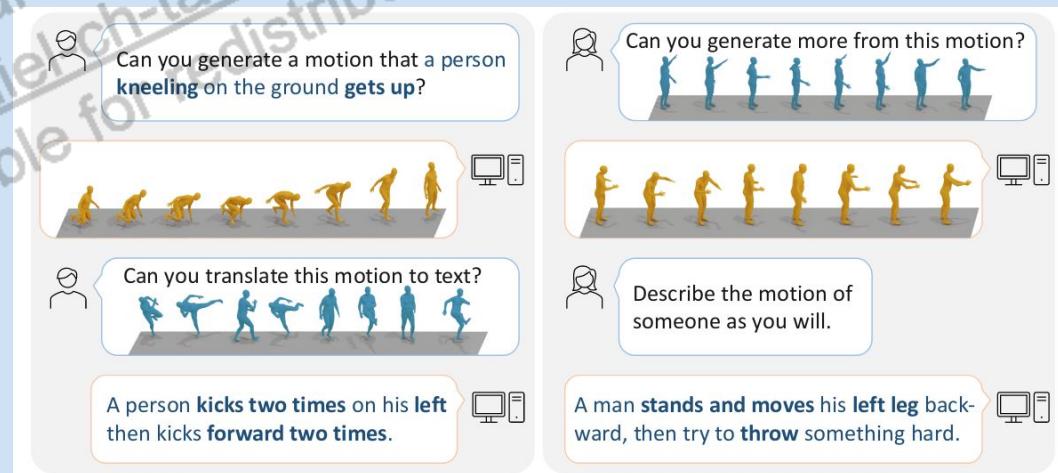
This: Directly using LLM as
a policy

Robot actions: simply another modality

LLMs can simply be trained to directly output low-level actions



At test time, the LLM robustly answers complex motion-text queries



RT-2: Massively pre-trained LLM-as-policy

Pre-train on internet
+ robot data

Internet-Scale VQA + Robot Action Data

Q: What is happening in the image?
A: 311 423 170 55 244

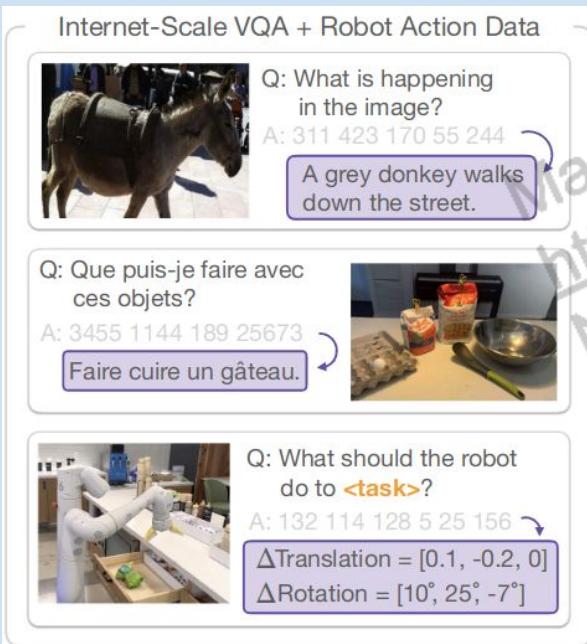
A grey donkey walks down the street.

Q: Que puis-je faire avec ces objets?
A: 3455 1144 189 25673

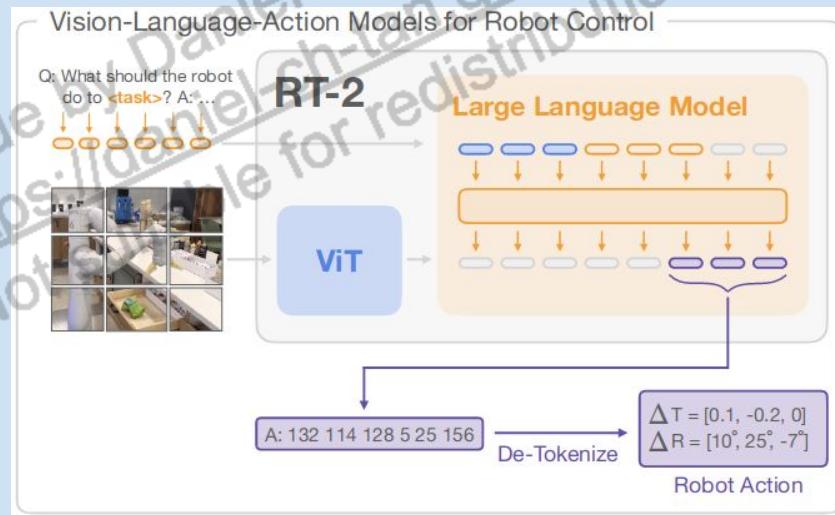
Faire cuire un gâteau.

Q: What should the robot do to <task>?
A: 132 114 128 5 25 156

$\Delta\text{Translation} = [0.1, -0.2, 0]$
 $\Delta\text{Rotation} = [10^\circ, 25^\circ, -7^\circ]$



Fine-tune end-to-end to directly output robot actions



Obtain generalizable, instruction-following policy



Solving complex manipulation-reasoning tasks



put strawberry into the correct bowl
pick up the bag about to fall off the table
move apple to Denver Nuggets
pick robot
place orange in matching bowl



move redbull can to H
move soccer ball to basketball
move banana to Germany
move cup to the wine bottle
pick animal with different colour



move coke can to Taylor Swift
move coke can to X
move bag to Google
move banana to the sum of two plus one
pick land animal

→ Visual reasoning
("bag about to fall off table")

→ Common sense
("land animal")

→ Arithmetic
("move banana to the sum of two plus one")

→ Prior knowledge
("Germany", "Taylor Swift")

Robust generalization capabilities



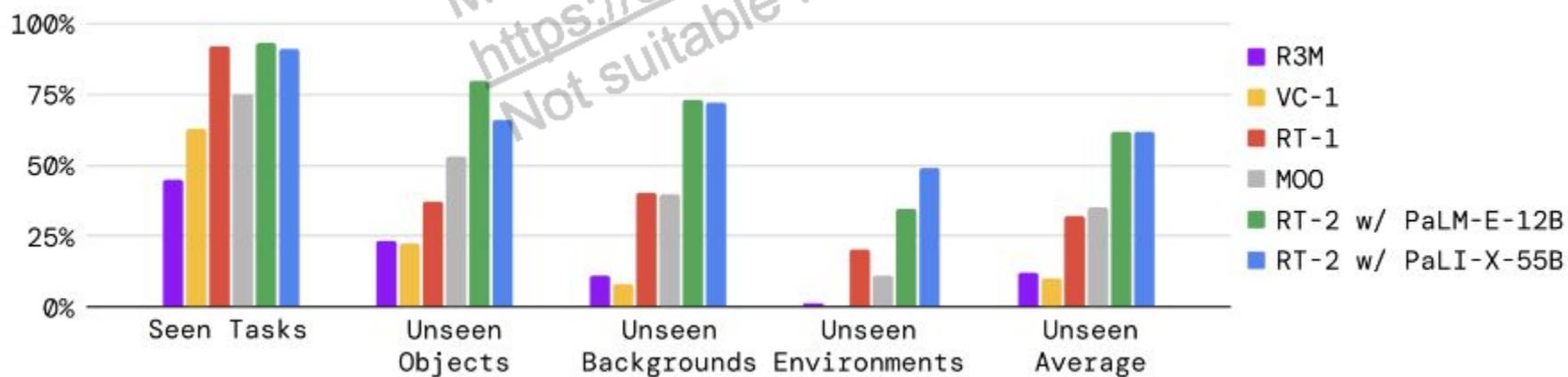
(a) Unseen Objects



(b) Unseen Backgrounds



(c) Unseen Environments

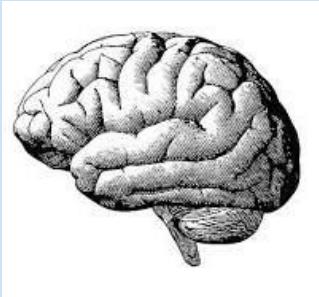


Takeaways here

- Robot actions are just another modality
- LLMs pre-trained on internet data can be fine-tuned as policies
 - Scale to tasks involving complex visual reasoning
 - Internet data enables massive generalization

Abstractness

Thoughts



Intentions



LLMs

- +: Text pre-training
- : Not grounded

Actions



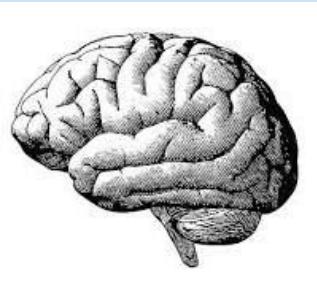
Robotics

- +: Grounded feedback
- : Hard to scale up

- 1. How to connect LLMs to robots for decision-making?**
- 2. How to scale bottom-up robot learning?**

Abstractness

Thoughts



Actions



Just train end to end!

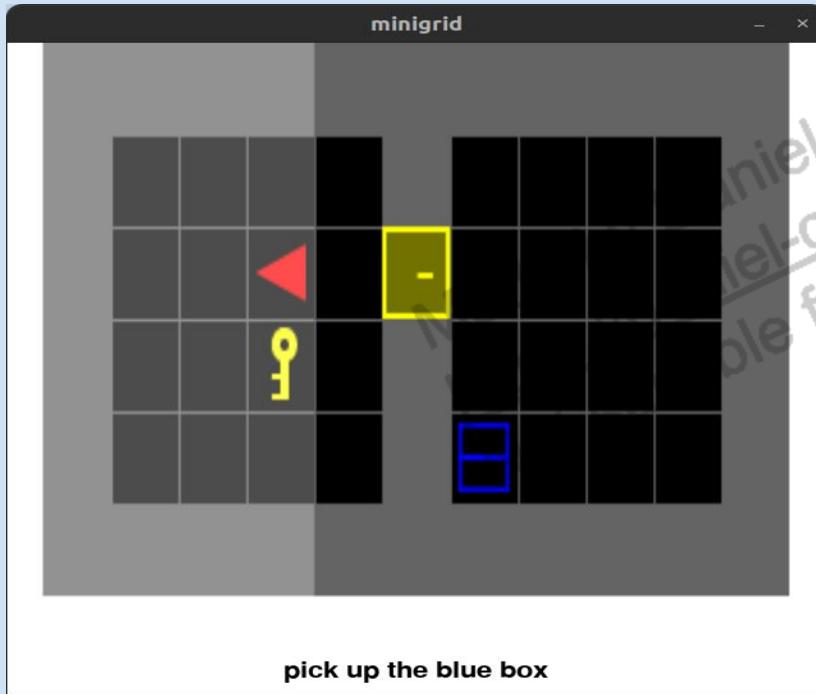
Made by Daniel C.H. Tan
<https://daniel-ch-tan.github.io/>
Not suitable for redistribution

Remainder of talk

1. How can LLMs accelerate robot learning?
2. **An end-to-end example in MiniGrid**
3. Future work

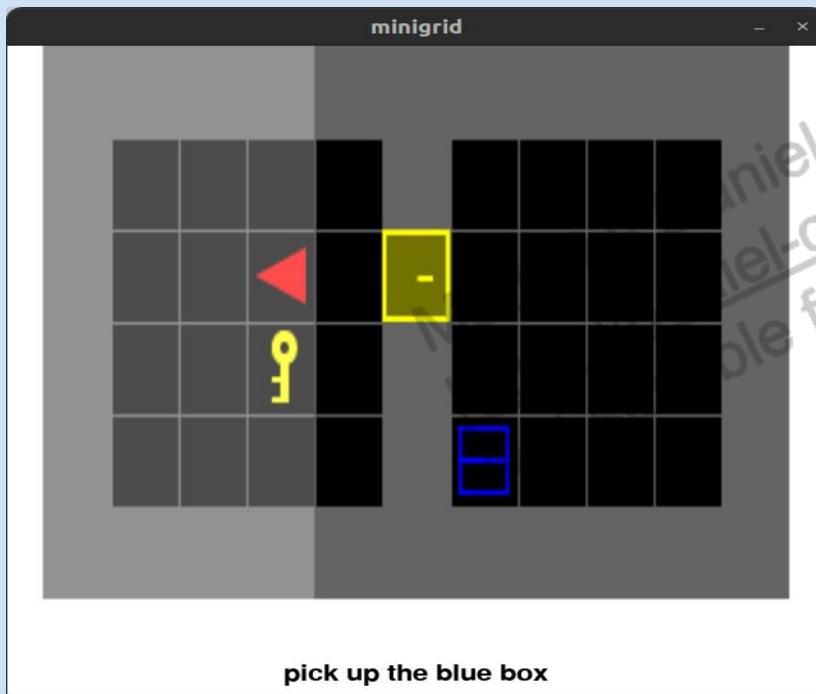
Made by Daniel C.H. Tan
<https://daniel-ch-tan.github.io/>
Not suitable for redistribution

The MiniGrid environment



Overall task:
Agent is carrying box

Baseline: Oracle Decomposition, Rewards



Overall task:

Agent is carrying box

Intermediate Objectives:

1. Agent is next to key
2. Agent is carrying key
3. Agent is next to door
4. Door is open
5. Agent is next to box
6. Agent is carrying box

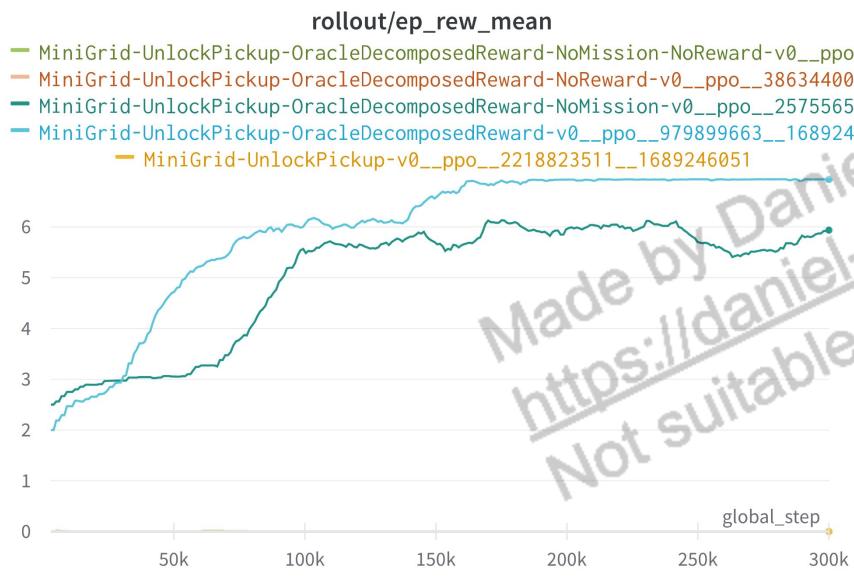
Goal conditioned
policies observe intermediate objective

Reward Functions

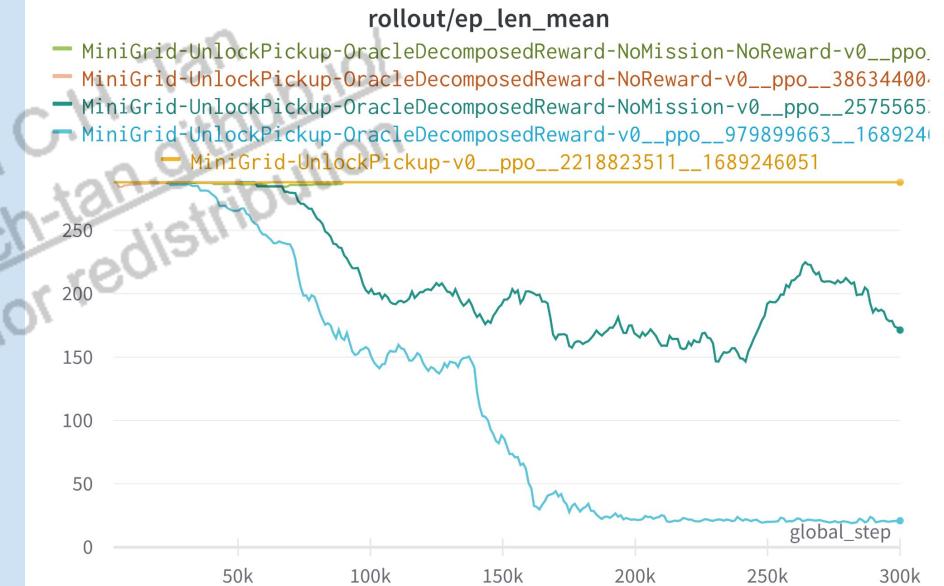
Manually coded

Reward shaped
policies get the shaped reward

Ablating goals, reward shaping



Only **reward-shaped** policies experience any positive reward



Goal-conditioning offers a small benefit by making the problem more Markovian

LLM-Decompose

Overall Task:

“red ball is next to
green key”

LLM-Evaluate

Intermediate
Objectives:

“Agent is next to
red ball”

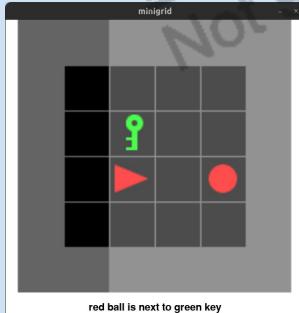
Reward Functions:

def

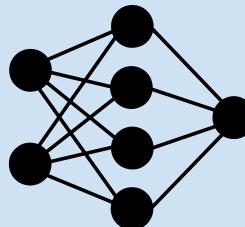
```
is_next_to_red_ball(obs)  
-> bool:
```

RL Loop

Environment



Policy



Shapes the
reward

LLM-Decompose

Prompt strategy:

1. **Describe MiniGrid**
2. **Describe current state**
3. Instructions
 - a. “Only use static information”
 - b. Formatting
4. Prompt-engineering:
 - a. Chain of Thought
 - b. Few-shot examples

I have an agent in a MiniGrid environment.

The agent can perform the following actions:

- Move forward
- Turn left
- Turn right
- Pick up an object (max 1 at a time)
- Place object (if currently holding one)
- Open / close a door

The environment is: \${environment}

The final state the agent must reach is: '\${mission}'.

Full prompt:

<https://pastebin.com/8Ly6ufKv>

LLM-Decompose

Prompt strategy:

1. **Describe MiniGrid**
2. **Describe current state**
3. **Instructions**
 - a. **“Only use static information”**
 - b. Formatting
4. **Prompt-engineering:**
 - a. **Chain of Thought**
 - b. Few-shot examples

Full prompt:

<https://pastebin.com/8Ly6ufKv>

Rules:

1. **Do this in the same style as 'red box is next to green ball'.**
2. **Do not use dynamic/action information in the states (e.g. velocities actions), only use static information.**
3. Always begin the list of intermediate states with <objectives> and end it with </objectives>

Follow this template:

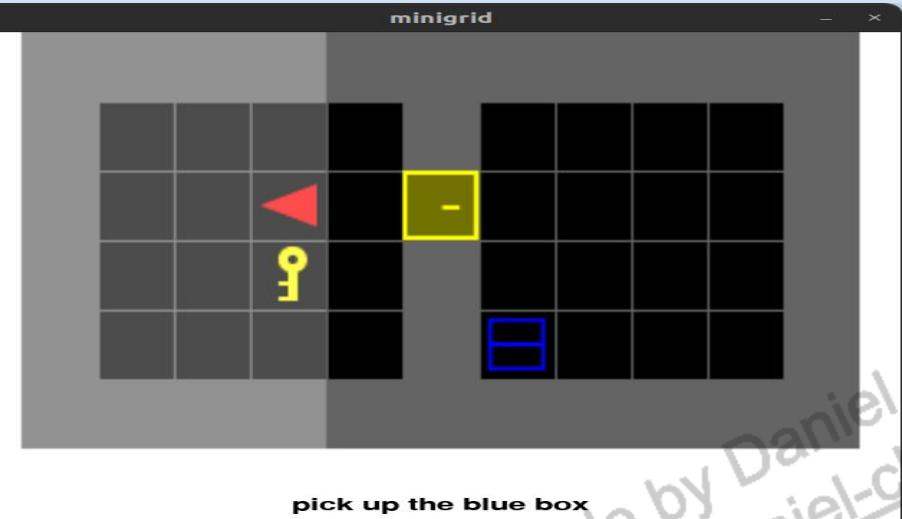
Thought: \${description of reasoning process}
(repeat above any number of times needed...)

The objectives are:

<objectives>
#. \${short description of intermediate state}
(repeat above any number of times needed...)
</objectives>

Example of intermediate state:

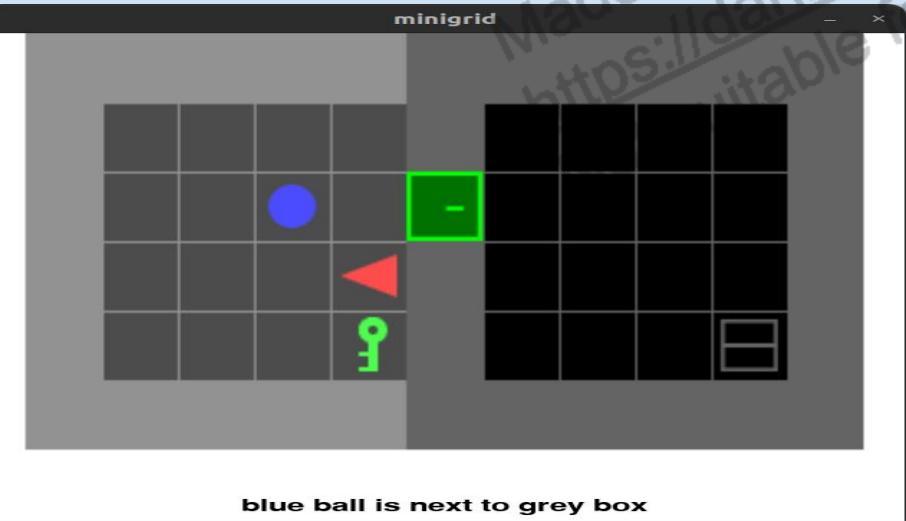
- 'Agent is next to grey door'
- 'Agent is carrying yellow key'
- 'Green door is open'



pick up the blue box

pick up the blue box

```
objective 0: 'Agent is carrying yellow key'  
objective 1: 'Agent is next to yellow door'  
objective 2: 'Yellow door is open'  
objective 3: 'Agent is in room (0, 1)'  
objective 4: 'Agent is not carrying any object'  
objective 5: 'Agent is next to blue box'  
objective 6: 'Agent is carrying blue box'
```



blue ball is next to grey box

blue ball is next to grey box

```
objective 0: Agent is next to green key  
objective 1: Agent is carrying green key  
objective 2: Agent is next to green door  
objective 3: Green door is open  
objective 4: Agent is in room (0, 1)  
objective 5: Green key is in room (0, 1)  
objective 6: Agent is in room (0, 0)  
objective 7: Agent is next to blue ball  
objective 8: Agent is carrying blue ball  
objective 9: Agent is in room (0, 1)  
objective 10: Blue ball is next to grey box
```

LLM-Evaluate

Prompt strategy:

1. Prepend decomposition context
2. Describe goal state
3. Instructions
4. Prompt-engineering:
 - a. Chain of Thought
 - b. Few-shot examples

Full prompt:

<https://pastebin.com/8Ly6ufKv>

[[Start from decomposition chat message history]]

Write a function that takes in the observation and checks whether state '\${state}' has been reached.

You should explain how the function works at a high-level, and any assumptions made, in a comment at the top of the function.

Rules:

1. Define exactly one function. Do not define extra helper functions.
 2. The function must be callable with a single argument, which is the observation.
 3. The function returns a boolean.
 4. Always enclose your code in a Python code block, e.g. ```python {your code here} ```
 5. Your function should be specific to the state described above.
 6. Do not change the function signature under any circumstances.
-

Here is an example observation.

...

Here is an example function for the state "agent is next to red box".

...



Sure, here is a Python function for that specific condition:



python

Copy code

```
def is_in_room_with_purple_box(observation):
    """Check if the agent is in the room with the purple

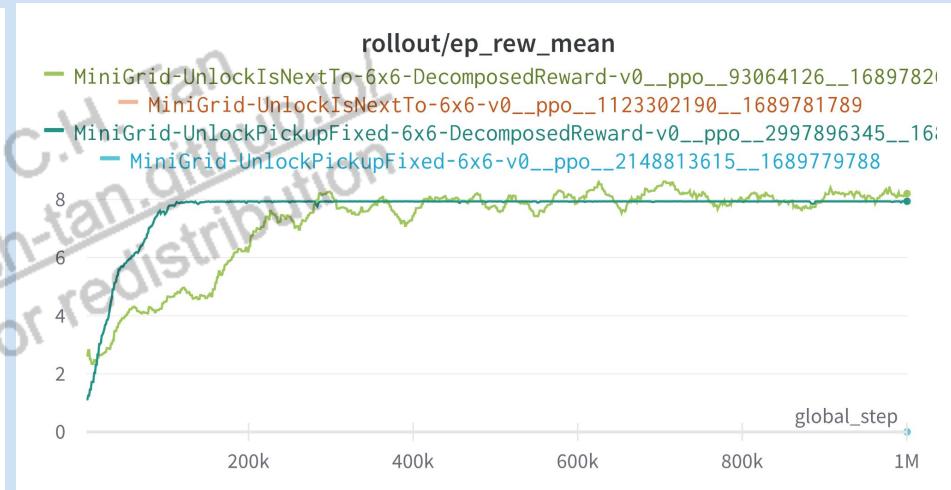
    if 'purple_box' in observation['field_of_view']:
        return True
    return False
```

This function also takes the agent's current observation as the argument. It will return `'True'` if the purple box is in the agent's field of view (implying that the agent is in the same room as the purple box), and `'False'` otherwise.

Evaluating on difficult environments



Success on 7-step environment
Failed to learn 11-step environment



11-step policy learns intermediate steps (up to 8th step)

Fails to learn to actually complete the task

Challenges

Require manually defining a perception pipeline

Relies on good zero-shot generation (of goals and checks).
→ Hard to do when env is stochastic, occluded

Success-checking functions generated are superficial

Remainder of talk

1. How can LLMs accelerate robot learning?
2. An end-to-end example in MiniGrid
3. **Future work**

Made by Daniel C.H. Tan
<https://daniel-ch-tan.github.io/>
Not suitable for redistribution

Strengths / weaknesses of LLMs in robotics:

Strengths:

- Semanticity; common sense
- A universal “glue” between modules
- Adapting to user feedback

Weaknesses:

- Hallucination → fundamental, but solvable by ‘outsourcing’ to PDDL, prob-prog, etc.
- Grounding → transient; future multi-modal LLMs could be pre-trained or fine-tuned on robot data

Robotics-oriented LLM evals

Focus on *embodied decision-making* capabilities

- **Datasets**, simulation-backed **benchmarks**
- Fine-tuned open-source **checkpoints**

Things to evaluate:

- Generate plans robust to perturbation (planner)
- Accurately evaluate a wide range of task completions (reward fn)
- Correctly predict future states (world model)
- Correctly predict robot actions (policy)

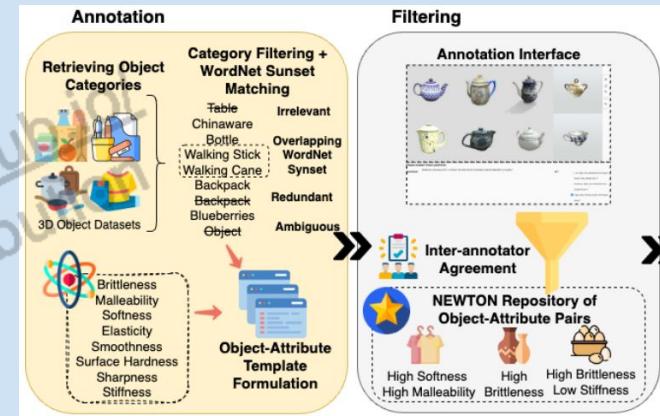
Reasoning about physical attributes

NEWTON: Everyday tasks
that require object
understanding

I need to prepare a work table as a play area for kids. Which of
scissors, pizza, cake, banana should I remove?



Generated from
ground-truth obj
datasets



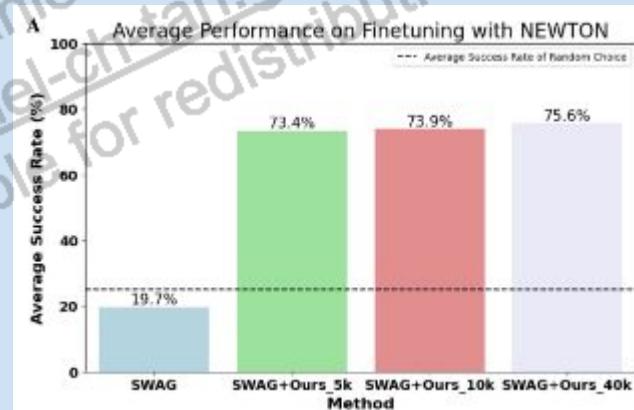
| Category | Template |
|--------------------|---|
| Elasticity | If you were to deform <MASK> (by squeezing, hitting, etc.) can it recover to its original form? a) No: Object will not be able to return to its original form. b) Somewhat: Object may recover to its original form but will have signs of previous deformations. c) Yes: Object is capable of recovering near perfectly to the original form. |
| Surface Smoothness | What is the surface smoothness of <MASK>? a) Low: majority of surface is very rough, or has many bumps. b) Moderate: some parts of surface are smooth, while some parts are not. c) High: majority of surface is smooth. |

Reasoning about physical attributes

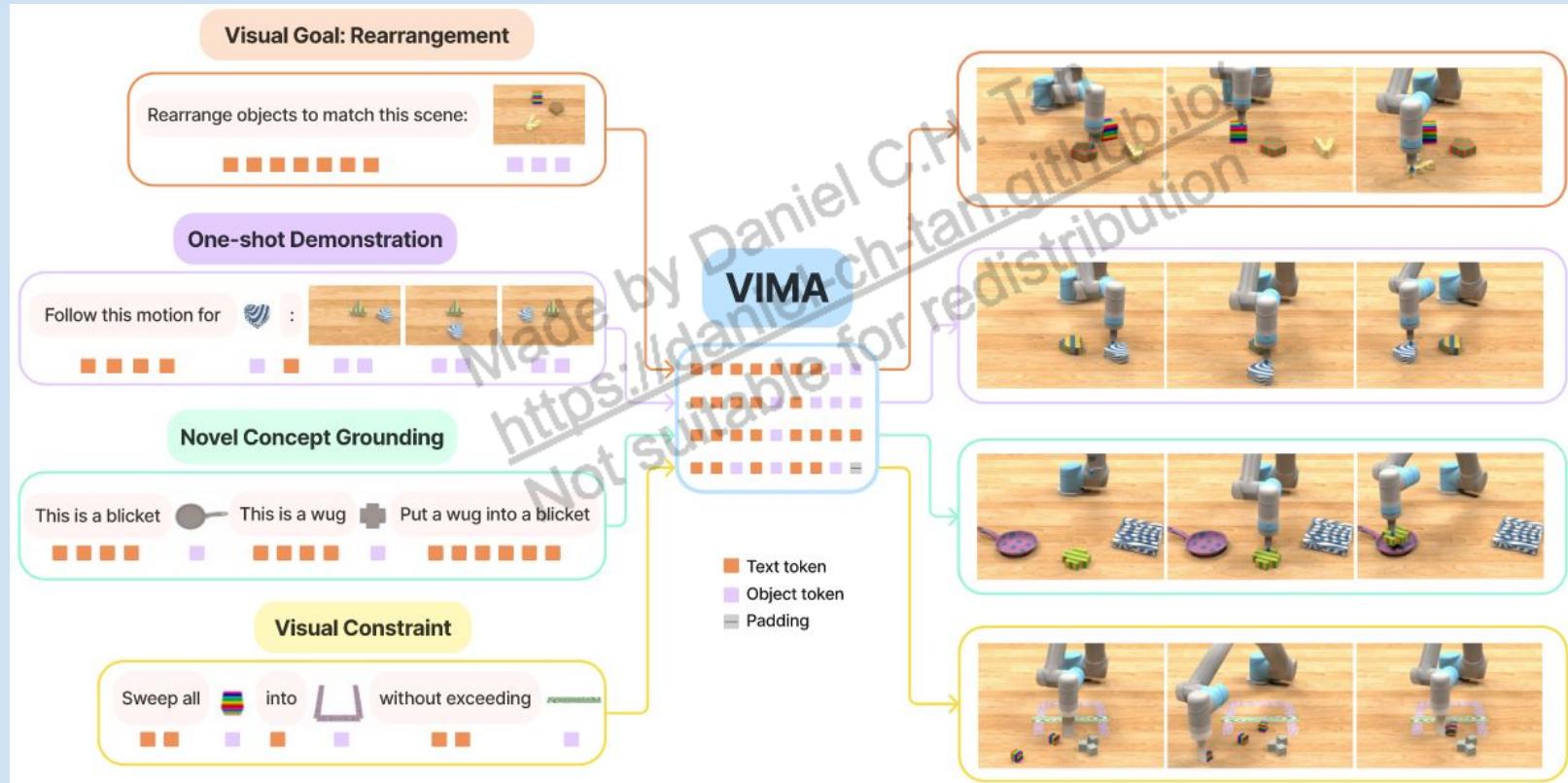
Zero-shot performance is lacking (~50%)

| | |
|-----------------------|-------------|
| Dolly-V2-7B | 7.6 |
| Flan-Alpaca-GPT4-XL | 15.2 |
| Flan-T5-small | 0.0 |
| Flan-T5-Base | 41.7 |
| Flan-T5-Large | 13.7 |
| Flan-T5-XL | 27.5 |
| UnifiedQA-V2-T5-Large | 14.3 |
| Alpaca-LoRa-7B | 10.0 |
| GPT-Turbo | 16.4 |
| GPT-4 | 49.7 |
| Average | 19.6 |
| Human | 84.4 |

Performance improves with fine-tuning!



Evaluating embodied decision-making



One-shot Demonstration

Follow this motion for :



:



VIMA



Novel Concept Grounding

This is a blicket



This is a wug



Put a wug into a blicket



Visual Constraint

Sweep all



into



without exceeding



- Text token
- Object token
- Padding



Trustworthiness

Can we apply ideas from mechanistic interpretability to understand what LLMs are thinking when they make embodied decisions?

Recent work shows LLMs learn emergent linear representations of:

- The Othello board state (OthelloGPT)
- Space and time
- Honesty (Discovering Latent Knowledge...)

Do LLMs learn linear representations of physical attributes? Goals? Feasibility?

Can we then control LLM behaviour by intervening on these representations?

RL with Environment Feedback

Note: RLHF has been rather effective in improving LLMs' performance on tasks of choice. Can we use the same machinery for learning from synthetic reward?

I.e. fine-tuning LLMs directly with environment feedback, in an RL sense

- Current SoTA LLM-as-policy approaches only do imitation learning! RL is more powerful (theoretically)
- In robotics environments, we have an *analytic* task reward → avoid issues with reward mis-specification
- Still reap the generalization benefits of having pre-trained on diverse internet data

Overall takeaways

- LLMs: a powerful source of semanticity
- LLMs improve many parts of the robot learning pipeline
- “Just train end-to-end” for connecting LLMs to robotics?
- We need better evals for LLMs in this space!

Thank you!

Please give me feedback!

<http://bit.ly/daniel-tan-talk-feedback>



© 2018 Daniel C.H. Tan
<https://daniel-ch-tan.github.io/>
Not suitable for redistribution