Designing Skills With LLMs

Martin Klissarov, Pierluca D'oro, Pierre-Luc Bacon,
Amy Zhang, Mikael Henaff, ...

Donghu Kim

Introduction

How does reinforcement learning work? → **Maximizing rewards.**

e.g., Training a roomba with RL → Clean room = More reward

What is the underlying philosophy of these rewards? → **Human Preference.**

e.g., Obviously, clean room > dirty room

Problem: **Reward design takes way too much effort.**

- Rewards are hard to define (Task specification problem)

  e.g., What is a clean room? What should be considered ‘dirty’?

- Rewards are hard to make ‘dense’ enough (Reward hacking problem)

  e.g., Reward = Amount of dust sucked in → What will happen?

[1] The Task Specification Problem., Pulkit Agrawal

Introduction

If only there was something that can automatically decide what humans prefer…

- LLMs / VLMs have learned human common sense from the internet!
- **Motif**: Let's use LLMs' common sense to design rewards!
- **MaestroMotif**: Let's use LLMs' common sense to design multiple types of rewards (i.e., skills)!

[1] Motif: Intrinsic Motivation from Artificial Intelligence Feedback., Klissarov et al.
[2] MaestroMotif: Skill Design from Artificial Intelligence Feedback., Klissarov et al.

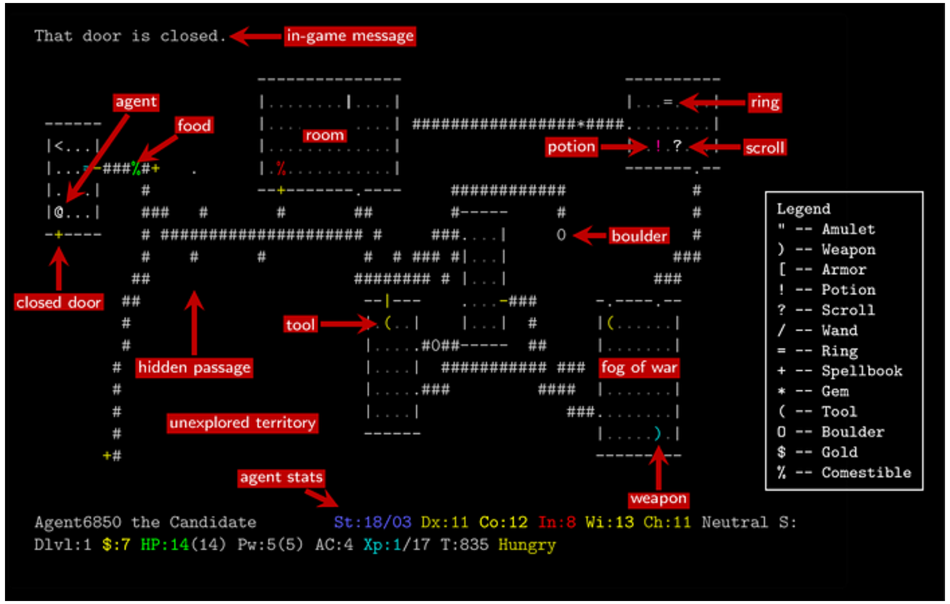
But first…

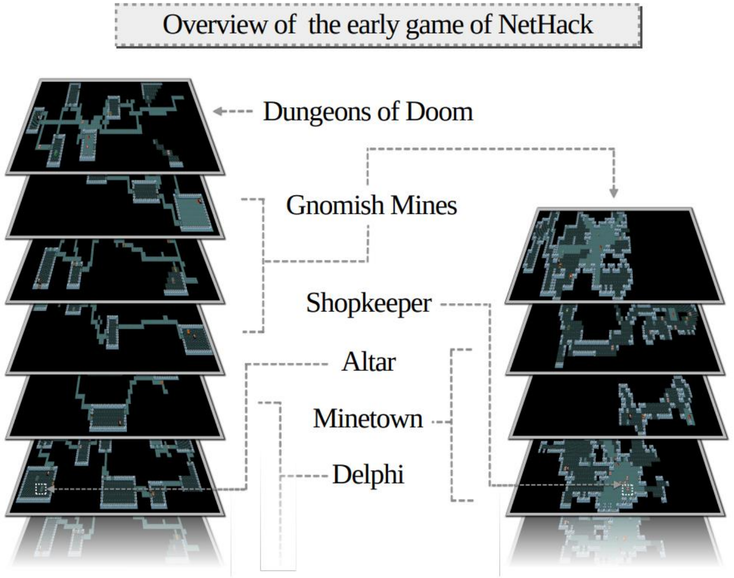

Welcome to NetHack
NetHack 3.6.4
NetHack, Copyright 1985-2019
By Stichting Mathematisch Centrum and M. Stephenson.
Version 3.6.4-0 Windows post-release, built Dec 21 20:09:43 2019.
See license for details.
No news.
OK

NetHack

What is NetHack?

- A text-based rougelike game first released in 1987. (Rougelike = You die, everything resets)
- Goal: Go through the *Mazes of Menace* and find the *Amulet of Yendor* to gain immortality (ascend).
- You will: discover paths, fight monsters, eat when hungry, collect money/weapon, disable traps, …



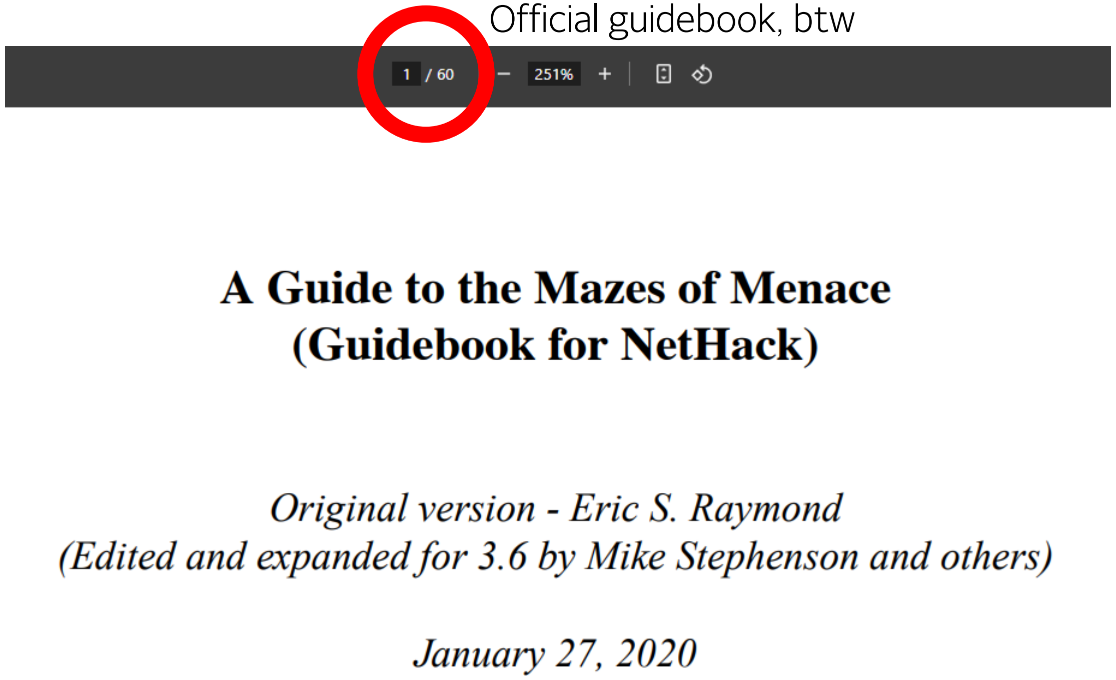


NetHack

What is NetHack?

- ...it is ***excruciatingly*** difficult.
- ”I played off and on for almost twenty years without ascending.”
  - A dude on Reddit, followed by another comment saying “Same.”
- Requires a lot of knowledge about the game.

Official guidebook, btw


1 / 60 251%

A Guide to the Mazes of Menace
(Guidebook for NetHack)

Original version - Eric S. Raymond
(Edited and expanded for 3.6 by Mike Stephenson and others)

January 27, 2020

NetHack

Why NetHack?

- The sheer complexity and diversity opens a huge potential for RL research.

based roguelike game, NetHack. We argue that NetHack is sufficiently complex to drive long-term research on problems such as exploration, planning, skill acquisition, and language-conditioned RL, while dramatically reducing the computational resources required to gather a large amount of experience. We compare NLE and

- Extremely fast simulation
- Game states are often labelled in text (in 'message' form)
- Huge human play dataset available

Lastly, there is also a large public repository of human replay data (over five million games) hosted on the NetHack Alt.org (NAO) servers, with hundreds of finished games per day on average [47].

- LLMs already have some knowledge about the game

What do you know about NetHack and the goal of a player in it? (Llama 2 70b)

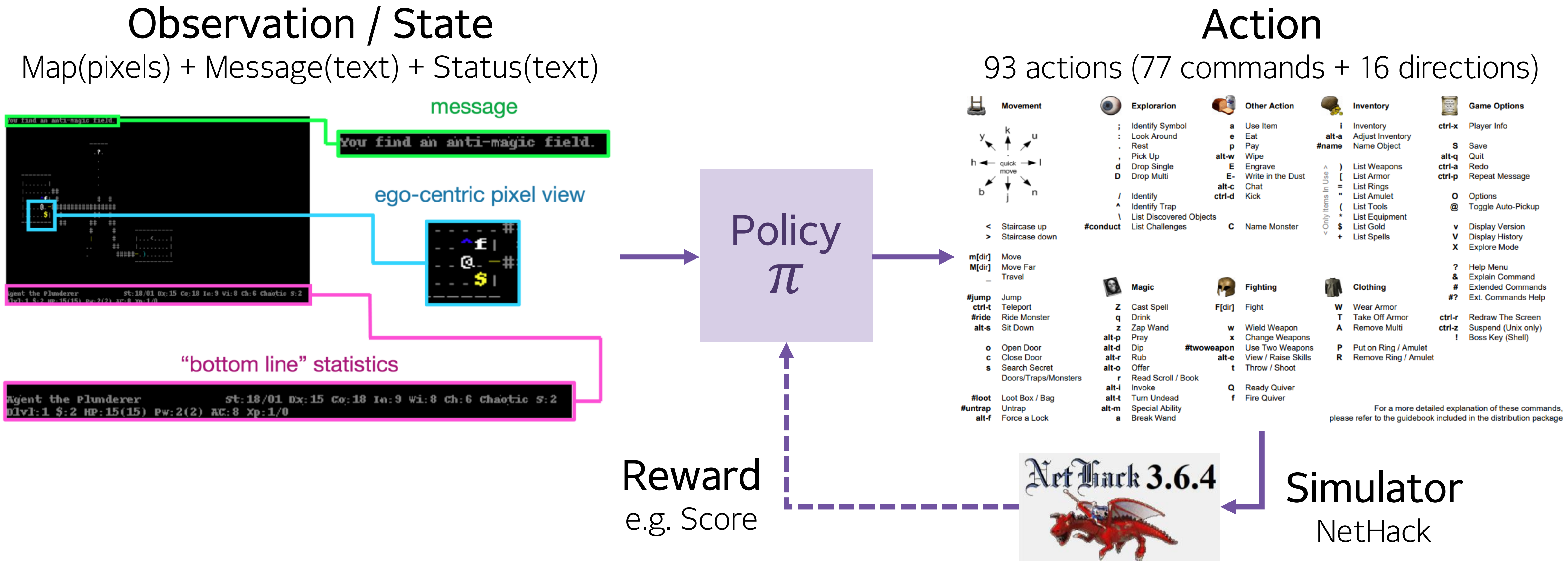
```
NetHack is a popular open-source roguelike video game that was first
released in 1987.  It is a descendant of the 1980 game Rogue, and
it has been widely influential in the development of the roguelike
genre.  The game takes place in a procedurally generated dungeon,
where the player controls a character known as the "hero" who must
navigate through levels filled with monsters, traps, and treasure.

The ultimate goal of the player in NetHack is to find and retrieve
the Amulet of Yendor, which is located on the lowest level of the
```

[1] The NetHack Learning Environment., Kuttler et al.

NetHack Learning Environment

The NetHack Learning Environment (NLE)

- There are other configurations, but we'll talk about the one used in Motif and MaestroMotif.



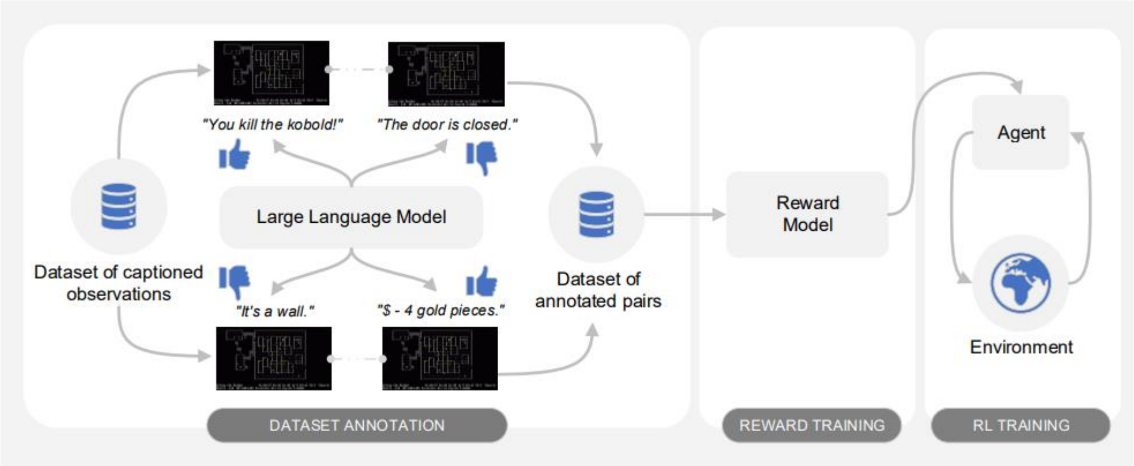
[1] NetHack is Hard to Hack., Piterbarg et al.

Motif: LLM-assisted Reward Design

Let's make LLMs represent human preference / common sense

- Method resembles RLHF (but now using LLM preference to train RL policies)

1. Prepare a bunch of observations with caption.
2. Randomly sample 2 observations, ask LLM which is 'better' based on its caption (win/lose/draw).
3. Repeat that K times to get preference dataset D_{pref}.
4. Train a reward model r_ϕ to predict LLM's preference.
5. Use the reward from r_ϕ to train an RL agent!



Motif: LLM-assisted Reward Design

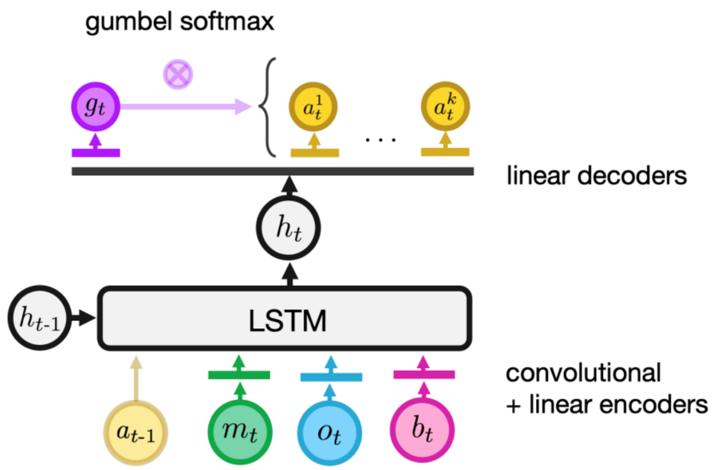
1. Prepare a bunch of observations with caption.
   - NetHack already does the captioning for us
     e.g., “You killed [monster]!” (positive), “You are starving” (negative), “You bump into a wall” (neutral)

   - Data collection: Used prior RL method (CDGPT5) to collect roughly 10M observations

To construct a reasonably diverse dataset $\mathcal{D}$, we collect a set of 100 episodes at every 100 million steps of learning with the standard NLE RL baseline CDGPT5 (Miffyli, 2022) and repeat the process for 10 seeds. The CDGPT5 baseline is trained for 1 billion steps to maximize the in-game score. We analyze these choices in Appendix H.3.

   - CDGPT5 = LSTM + PPO (asynchronous) + BC

The `CDGPT5` model consists of three separate encoders: a 2-D convolutional encoder for pixel-rendered visual observations of the dungeon o_t, a multilayer perceptron (MLP) encoder for the environment message m_t, and a 1-D convolutional encoder for the bottom-line agent statistics b_t.



[1] NetHack is Hard to Hack., Piterbarg et al.

Motif: LLM-assisted Reward Design

2. Randomly sample 2 observations, ask LLM which is 'better' based on its caption
3. Repeat that K times to get preference dataset D_{pref}.
   - 1 if {message_1} is better, 2 if the other, $\emptyset$ if draw
   - Automatic draw if two messages are same (happens 5~10%)
   - LLaMA-2-70B + Chain-of-Thought + Paged Attention
   - Total 500K preference pairs from 10M dataset.

System prompt

```
You are a helpful and honest judge of good gameplaying and progress
in the NetHack game.  Always answer as helpfully as possible, while
being truthful.

If you don't know the answer to a question, please don't share false
information.
```

Zero-knowledge prompt

```
I will present you with two short gameplay descriptions.  First, tell
me about your knowledge of NetHack.  Mention the goal of NetHack.
{modifier}

Then, write an analysis describing the semantics of each description
strictly using information from the descriptions (which may be empty)
and your knowledge of NetHack.  Provide a comparative analysis based
on first principcples.

Finally, respond by explicitly declaring which one is the most
likely to make some progress towards the goal, writing either
("best_description":  1), ("best_description":  2).  You could also
say ("best_description":  None).
{ "description_1":  "{message 1}" }
{ "description_2":  "{message 2}" }
```

Prompt modifiers

```
{default}:  "Prefer agents that maximize the score in the game, for
instance by killing monsters, collecting gold or going down the
stairs in the dungeon."
```

Motif: LLM-assisted Reward Design

2. Randomly sample 2 observations, ask LLM which is 'better' based on its caption
3. Repeat that K times to get preference dataset D_{pref}.

- Output example

"There is a staircase up here. You see here an orcish helm."
vs
"It's solid stone."

```
I have extensive knowledge of NetHack and its mechanics.  The goal of
NetHack is to descend through the dungeon levels, collecting treasure
and killing monsters, while avoiding death and reaching the Amulet
of Yendor, which is located on the bottom level.  The game is won by
taking the Amulet to the surface.

Description 1 mentions a staircase up and an orcish helm.  The
presence of a staircase suggests the possibility of descending
deeper into the dungeon, which aligns with the goal of the game.
The orcish helm could potentially provide a bonus to the player's
combat abilities, which would aid in killing monsters and collecting
treasure.

Description 2 simply states that the area is solid stone.  This
does not provide any information about the possibility of descending
deeper into the dungeon or encountering monsters or treasure.

Based on the information provided, Description 1 is more likely
to lead to progress towards the goal of the game.  Therefore,
("best_description":  1).
```

Motif: LLM-assisted Reward Design

4. Train a reward model r_ϕ to predict LLM's preference
   - Sample a pair from D_{pref}, train with 'standard loss function in preference-based RL'.

$$\mathcal{L}(\phi) = -\mathbb{E}_{(o_1,o_2,y)\sim\mathcal{D}_{\text{pref}}}\left[\mathbb{1}[y=1]\log P_\phi[o_1 \succ o_2] + \mathbb{1}[y=2]\log P_\phi[o_2 \succ o_1] + \mathbb{1}[y=\varnothing]\log\left(\sqrt{P_\phi[o_1 \succ o_2]\cdot P_\phi[o_2 \succ o_1]}\right)\right]$$

$$\text{where } P_\phi[o_a \succ o_b] = \frac{e^{r_\phi(o_a)}}{e^{r_\phi(o_a)} + e^{r_\phi(o_b)}}$$

   - $r_\phi(o)$ learns to predict the logits that matches the preference hierarchy between observations.

     i.e., higher logit to 'winning' observations, equal logits when observations are 'draw'.

   - Motif uses only the message (i.e., $r_\phi(m)$) with 1D convolution based architecture.

Equation 1 by gradient descent. For simplicity, we only use the `message` as the part of the observation given to this reward function, and process it through the default character-level one-dimensional convolutional network used in previous work (Henaff et al., 2022). To make it more amenable to RL

Motif: LLM-assisted Reward Design

5. Use the reward from r_ϕ to train an RL agent
   - Three post-processing techniques on r_ϕ
     1. Noise reduction: Remove any reward signal smaller than ϵ.
     2. Reward normalization: Subtract mean and divide std (computed from dataset) on every reward.
     3. Count-based normalization: Exponentially decay rewards from the same source (message).

(Prevents the agent from getting fixated in a single reward signal)

$$r_{\text{int}}(\mathtt{message}) = \mathbb{1}[r_\phi(\mathtt{message}) \geq \epsilon] \cdot r_\phi(\mathtt{message})/N(\mathtt{message})^\beta$$

* 'int' as in intrinsic reward (from the agent itself), in contrast to extrinsic reward (from the environment)

- Final reward: $r_{effective} = \alpha_1 r_{int} + \alpha_2 r_{ext}$.
- Train algorithm: CDGPT5 + PPO, for 2B steps.

Extrinsic Reward Coeff (score)	0.1	
Extrinsic Reward Coeff (others)	10.0	
Intrinsic Reward Coeff	0.1	
ϵ threshold	0.5 quantile	(i.e., use only top 50% of rewards)
β exponent	3	

Motif: LLM-assisted Reward Design

Experimental Setting

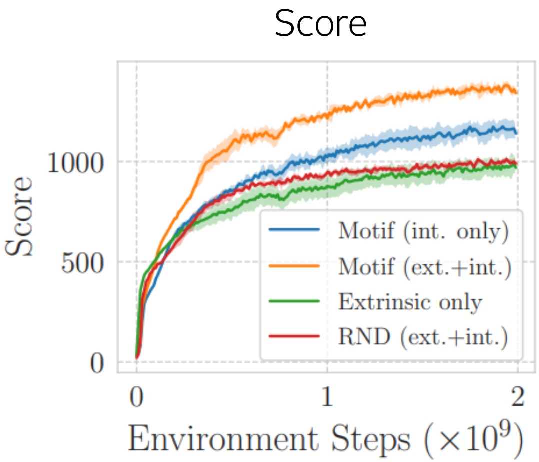
- Tasks (Environments)
  - **Score**: Maximize score (native in NetHack). [Extrinsic reward = Δscore]
  - **Staircase (level 2,3,4)**: Reach level X. [Extrinsic reward = 50 upon reaching level X]
  - **Oracle**: Find the character 'Oracle' (level>4) [Extrinsic reward = 50 upon finding Oracle]
- Baselines
  - No exploration **(Extrinsic only)**
  - Prior exploration method **(RND)**: Gives intrinsic rewards to novel states (haven't seen before).
  - Using LLM directly as policy **(Voyager)**: LLM generates code to create progressively more complex skills

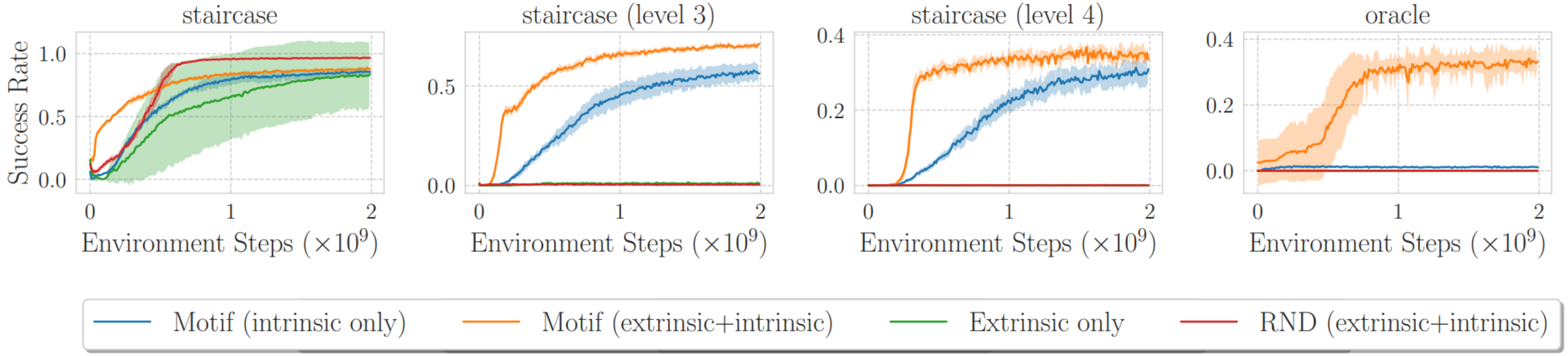
[1] Exploration by Random Network Distillation., Burda et al.
[2] Voyager: An Open-Ended Embodied Agent with Large Language Models., Wang et al.

Motif: LLM-assisted Reward Design

Experimental Setting & Quantitative Results

- Tasks (Environments)
  - **Score**: Maximize score (native in NetHack). [Extrinsic reward = Δscore]
  - **Staircase (level 2,3,4)**: Reach level X. [Extrinsic reward = 50 upon reaching level X]
  - **Oracle**: Find the character 'Oracle' (level>4) [Extrinsic reward = 50 upon finding Oracle]



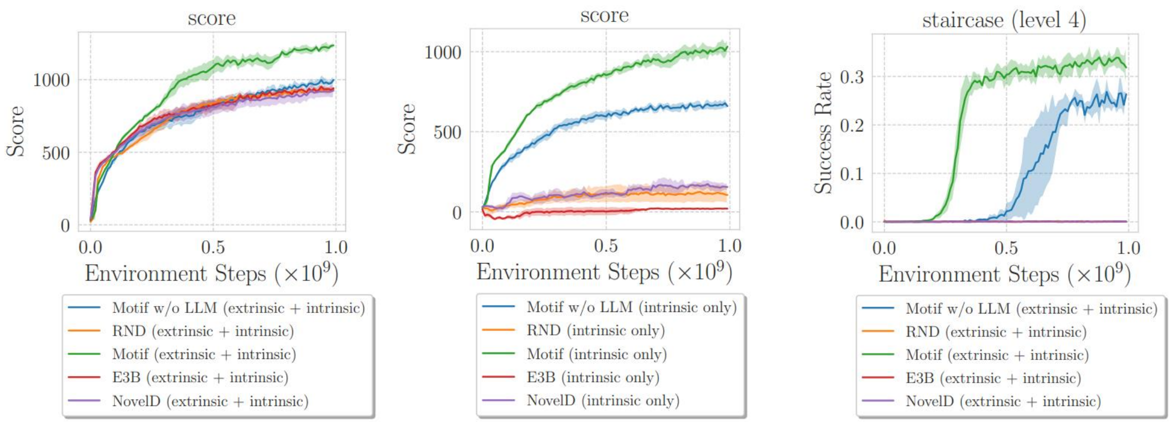


Not sure if RND is the right baseline…

Motif: LLM-assisted Reward Design

Quantitative Results

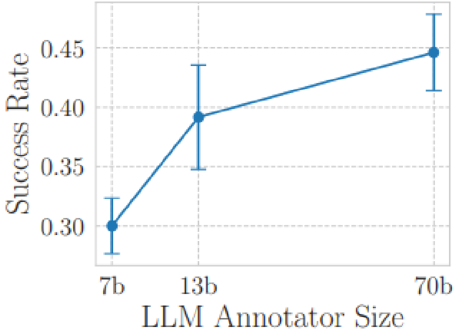
- Motif w/o LLM: $r_{int} = 1$ on every single message
    - Performs surprisingly well → Preference-agnostic exploration is still better than no exploration
    - Still, meaningful performance gap is made only with Motif (left) → Preference labelling is significant!
- Using LLM directly as policy (Voyager)
    - Didn't make progress → LLMs have common-sense, but not the ability to interact with low-level actions.



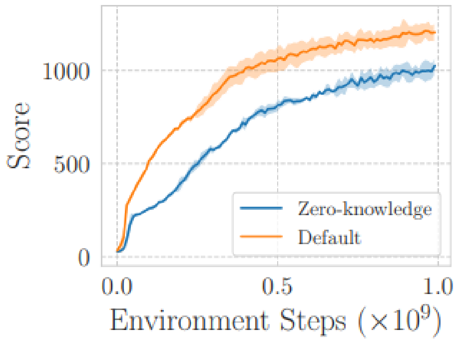
Motif: LLM-assisted Reward Design

Quantitative Results

- (a) Obviously, scales with LLM size.
- (b) Also obviously, better prompt yields better result.



(a) Scaling profile observed in the `staircase` `(level 3)` task.



(b) Effect of additional prompt information in the `score` task.

Zero-knowledge prompt

```
I will present you with two short gameplay descriptions.  First, tell
me about your knowledge of NetHack.  Mention the goal of NetHack.
{modifier}

Then, write an analysis describing the semantics of each description
strictly using information from the descriptions (which may be empty)
and your knowledge of NetHack.  Provide a comparative analysis based
on first principcples.

Finally, respond by explicitly declaring which one is the most
likely to make some progress towards the goal, writing either
("best_description":  1), ("best_description":  2).  You could also
say ("best_description":  None).
{ "description_1":  "{message 1}" }
{ "description_2":  "{message 2}" }
```

Prompt modifiers

```
{default}:  "Prefer agents that maximize the score in the game, for
instance by killing monsters, collecting gold or going down the
stairs in the dungeon."
```

Motif: LLM-assisted Reward Design

Analysis: Why is Motif good?

- Because the rewards made by LLM “align with human intuition”.
- Prefers messages like “The door opens”, which means that LLMs···
  1. **Exploration**: Have the natural tendency to explore the environment.
  2. **Credit Assignment**: Knows which messages lead to exploration/discovery and *directly* rewards it.
- Less likely to kill their pets

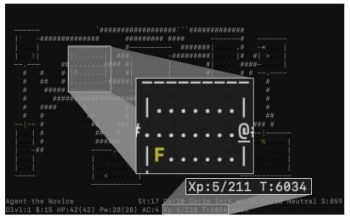
Highly preferred messages

```
The door opens.
With great effort you move the boulder.
You descend the stairs.
You find a hidden door.
You kill the cave spider!
As you kick the door, it crashes open!
You kill the newt!
You kill the grid bug!
You find a hidden passage.
You see here a runed dagger.
You hear the footsteps of a guard on patrol.  It’s a wall.
```

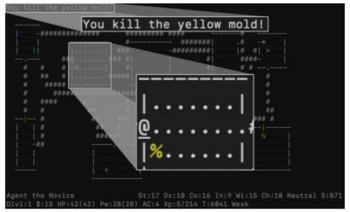
Motif: LLM-assisted Reward Design

Analysis: Reward hacking

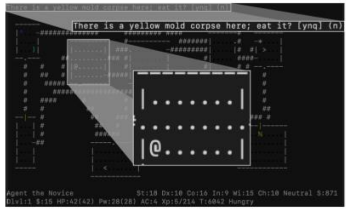
- Reward hacking can occur even with (or *because* of) dense rewards made by LLM.
- Hilarious example on Oracle task
  - Agent learned to drug itself and hallucinate a monster as the Oracle (why is this even in the game)
  - Didn’t happen with different LLM prompt. → Prompt sensitivity?



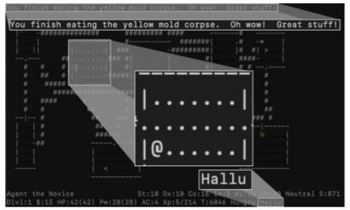
(1) Agent @ sees the monster F



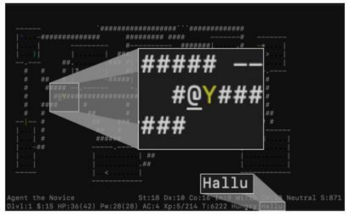
(2) Agent @ kills the monster



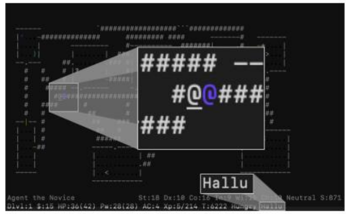
(3) Agent @ eats the corpse %



(4) Agent @ starts hallucinating



(5) A monster Y nears agent @



(6) The oracle @ is hallucinated

Motif: LLM-assisted Reward Design

Analysis: Steerability

- What if we tell the LLM to prefer specific characteristic?
- LLMs *do* reflect this in their preference,
  → which affects the reward signal,
  → which affects RL agent's behavior!
- **LLMs' preferences are steerable, and thus the RL agent!**

Agent	*The Gold Collector*	*The Descender*	*The Monster Slayer*
Prompt Modifier	Prefer agents that maximize their gold	Prefer agents that go down the dungeon	Prefer agents that engage in combat
Improvement	+106% more gold (64%, 157%)	+17% more descents (9%, 26%)	+150% more kills (140%, 161%)
	"In what direction?" "$ - 2 gold pieces." "$ - 4 gold pieces."	"In what direction?" "The door resists!" "You can see again."	"You hit the newt." "You miss the newt." "You see here a jackal corpse."

Zero-knowledge prompt

```
I will present you with two short gameplay descriptions. First, tell
me about your knowledge of NetHack. Mention the goal of NetHack.
{modifier}

Then, write an analysis describing the semantics of each description
strictly using information from the descriptions (which may be empty)
and your knowledge of NetHack. Provide a comparative analysis based
on first princicples.

Finally, respond by explicitly declaring which one is the most
likely to make some progress towards the goal, writing either
("best_description": 1), ("best_description": 2). You could also
say ("best_description": None).
{ "description_1": "{message 1}" }
{ "description_2": "{message 2}" }
```

Prompt modifiers

```
{default}: "Prefer agents that maximize the score in the game, for
instance by killing monsters, collecting gold or going down the
stairs in the dungeon."

{gold}: "Prefer agents that maximize their gold. But never prefer
agents that maximize the score in other ways (e.g., by engaging in
combat or killing monsters) or that go down the dungeon."

{stairs}: "Prefer agents that go down the dungeon as much as
possible. But never prefer agents that maximize the score (e.g.,
by engaging in combat) or that collect ANY gold."

{combat}: "Prefer agents that engage in combat, for instance by
killing monsters. But never prefer agents that collect ANY gold or
that go down the dungeon."
```

MaestroMotif: LLM-assisted Skill Design

We just saw that LLM preference is steerable, and thus the RL agent behavior.

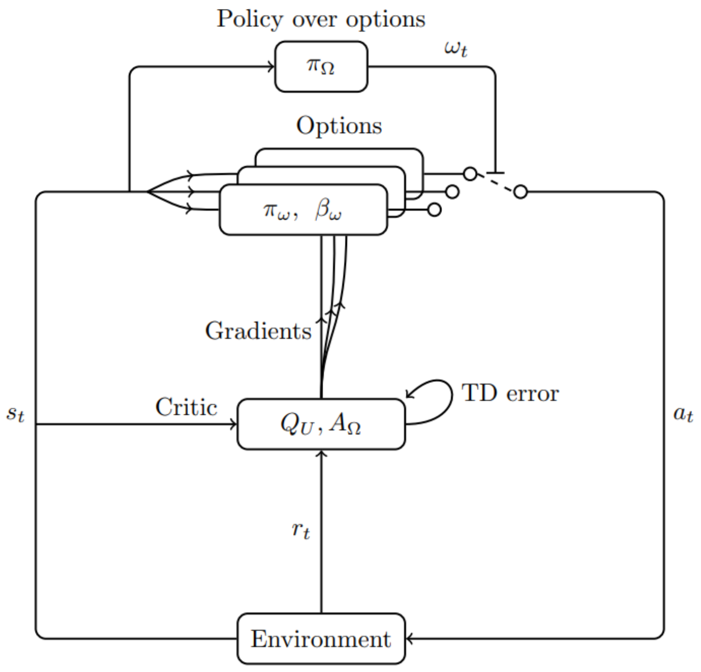
- This means that we can create Motifs to create distinct behaviors, i.e., skills!
  - Why skills? **Abstraction of action search space.**
  - For example, think of training a basketball humanoid.
    → Imagine all the possible actions (torques on each motor) that can be taken.
  - Us humans don't think of individual muscles when playing basketball.
    → Instead, we think of abstract concepts (skills) like run, pass, dribble, shoot.
  - Abstracting the action space greatly reduces the cost of searching/choosing good actions.
- All human has to do is give a high-level intuition of what each skill should be.

MaestroMotif: LLM-assisted Skill Design

1. Use Motif with different prompts to create distinct behaviors (skills).

- New problem: How do we combine multiple skills into one policy? How can we train them?
  - Now comes··· **Option Framework**

1. Each option (=skill) is a tuple (I_w, π_w, β_w)
   - Initiation (I_w): Determines when skill w can be triggered
   - Skill-policy (π_w): Policy of the skill
   - Termination (β_w): Determines when skill w must end
2. Policy over options (π_Ω) choose which skill to trigger in skillset Ω.



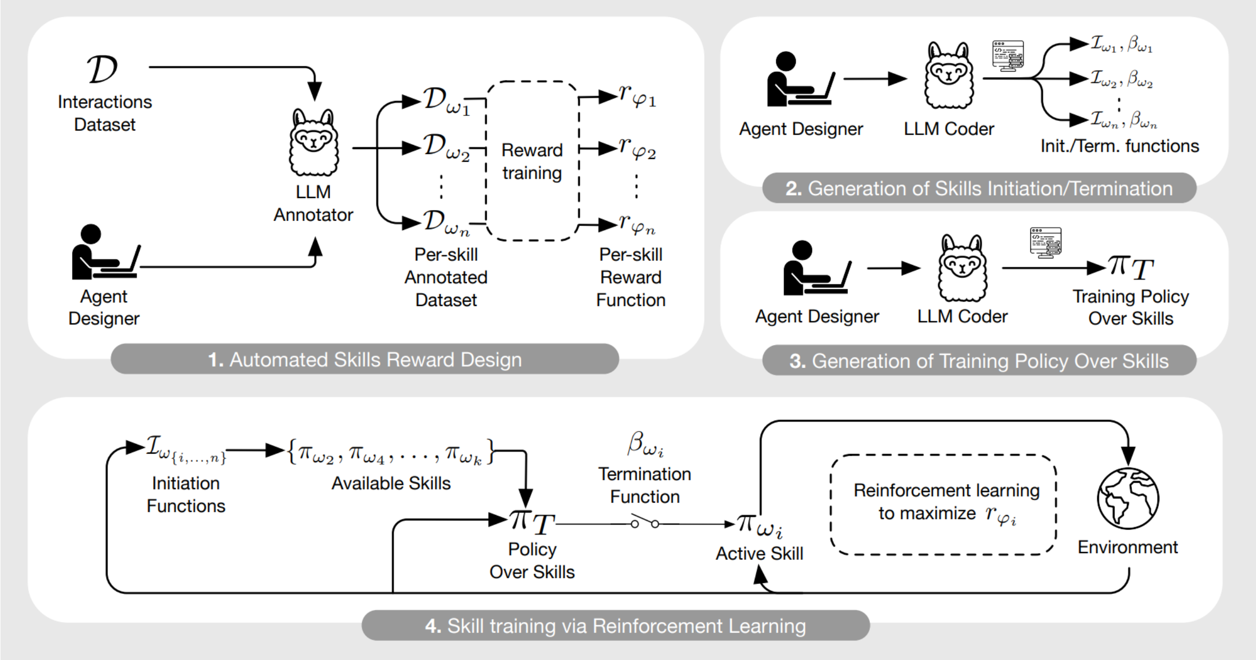
- Skill-policies (π_w) are what we learn with Motif.
- Fill the rest (I_w, β_w, π_Ω) with python codes generated by LLMs!

[1] The Option-Critic Architecture., Bacon et al.
[2] Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning., Sutton, Precup, Singh.

MaestroMotif: LLM-assisted Skill Design

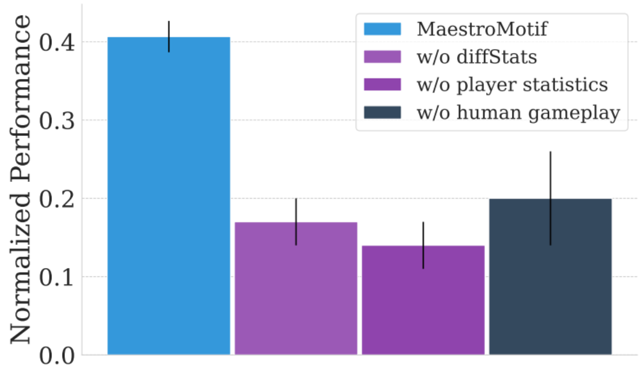
LLM-assisted skill design pipeline

1. Create preference datasets with different prompts asking for distinct behaviors (skills).
2. Make LLM generate codes for when to start/end each skill (I_w, β_w).
3. Make LLM generate code for policy-over-skills (π_T).
4. Train the complete set of options via RL.



MaestroMotif: LLM-assisted Skill Design

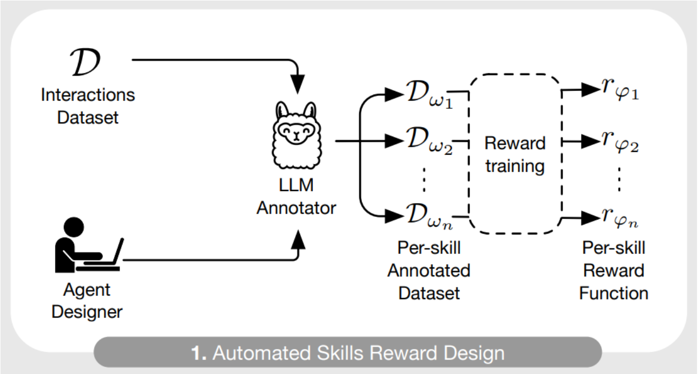
1. Create preference datasets with different prompts asking for distinct behaviors (skills).
   - Same pipeline as Motif, but ran with N distinct prompts.
     1. Prepare a bunch of observations with caption.
     2. Randomly sample 2 observations, ask LLM which is 'better' based on its caption (win/lose/draw).
     3. Repeat that K times to get preference dataset D_{pref}.
     4. Train a reward model r_ϕ to predict LLM's preference.
   - 3 crucial improvements from Motif
     - Improved dataset: RL agent dataset (Baseline Motif) + Human play dataset (Dungeons and Data)
     - More input to r_ϕ: messages + statistics (contains health, gold, level, ⋯)
     - \`diff\` state representation: See reference.



[1] diff History for Neural Language Agents., Piterbarg et al.

MaestroMotif: LLM-assisted Skill Design

1. Create preference datasets with different prompts asking for distinct behaviors (skills).
   - Generated skills for NetHack
     - Discoverer: Explore dungeon, collect items, survive.
     - Descender: Find staircases and go down a dungeon level.
     - Ascender: Find staircases and go up a dungeon level.
     - Merchant: Find a shopkeeper and interact with them.
     - Worshipper: Find an altar and interact with it (to identify whether items are cursed or not).



Prompt skill modifiers

{Discoverer}: "players that are adventurous but only within the same dungeon level, for example by fighting monsters, finding gold pieces or scrolls; but do not drop them. Categorically refuse going up and down dungeon levels."

{Descender}: "the direction of progress is to explore by going down the dungeon. It is urgent to do so, strongly avoid staying on the same level or worse, going higher."

{Ascender}: "the direction of progress is only by going up a dungeon level successfully. Strongly dislike remaining on the same dungeon level, no matter the consequences."

{Worshipper}: "strongly encourage players that interact with the altar, primarily for identifying whether items are cursed or blessed, rather than for praying to or pleasing their god."

{Merchant}: "prefer players that negotiate, sell and interact with shopkeepers. Be careful not to steal from stores."

MaestroMotif: LLM-assisted Skill Design

2. Make LLM generate codes for when to start/end each skill (I_w, β_w).
3. Make LLM generate code for policy-over-skills (π_T).

Prompt for the train-time policy over skills

```
You are to write code which defines the method "select_skill" of the
NetHack Player class that selects amongst a set of skills in the
videogame of NetHack.  The set of skills corresponds to {"discoverer",
"descender", "ascender", "merchant", "worshipper"}.

When activated, the Discoverer fully explores the current dungeon,
while fighting off enemies.  The Descender makes its way to a
staircase and goes down.  The Ascender makes its way to a staircase
and goes up.  The Merchant interacts with shopkeepers by selling
its items.  The Worshipper interacts with altars by identifying its
items.

Find a strategy that will let the player explore fully each of the
first few dungeon levels, alternating directions between going all
the way down towards the maximum depth, then going up towards the
first dungeon.  This might get interrupted by the end of the loop or
if the preconditions of worshipper and merchant allow for it.

You can keep track of any other information by assigning values to
other class attributes, but only if that really helps.

Your code will be verified through this unit test.

###
{unit_test}
###

Before writing the code, write a few questions for yourself, and
answer them to make sure you understand the assignment.
In your response make sure to include the code for the class
definition and the unit test in a python markdown.
```

Code for train-time policy over skills

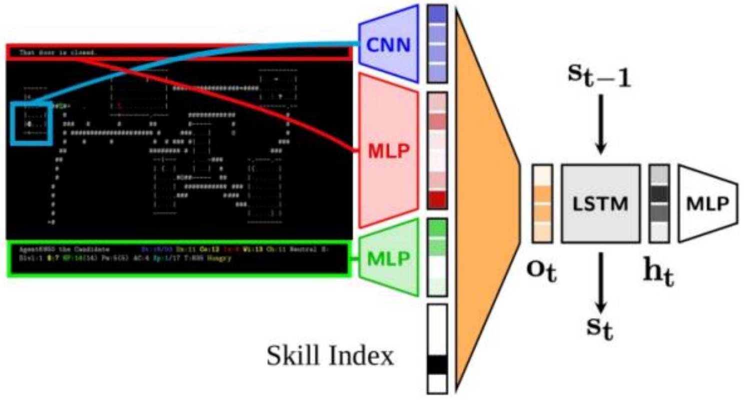
```
class NetHackPlayer:
    def __init__(self, max_depth, branch_depth):
        self.max_depth = max_depth
        self.branch_depth = branch_depth
        self.explored_levels = set()
        self.direction = 'down'  # Start by going down

    def merchant_precondition(self):
        # Placeholder for actual merchant precondition logic
        return False

    def worshipper_precondition(self):
        # Placeholder for actual worshipper precondition logic
        return False

    def select_skill(self, current_skill, dungeon_depth,
                     merchant_precondition, worshipper_precondition):
        if merchant_precondition:
            return 'merchant'
        if worshipper_precondition:
            return 'worshipper'

        if current_skill == 'discoverer':
            self.explored_levels.add(dungeon_depth)
            if self.direction == 'down':
                if dungeon_depth < self.max_depth:
                    return 'descender'
                else:
                    self.direction = 'up'
                    return 'ascender'
            elif self.direction == 'up':
                if dungeon_depth > 1:
                    return 'ascender'
                else:
                    self.direction = 'down'
                    return 'descender'
        elif current_skill == 'descender':
            return 'discoverer'
```

MaestroMotif: LLM-assisted Skill Design

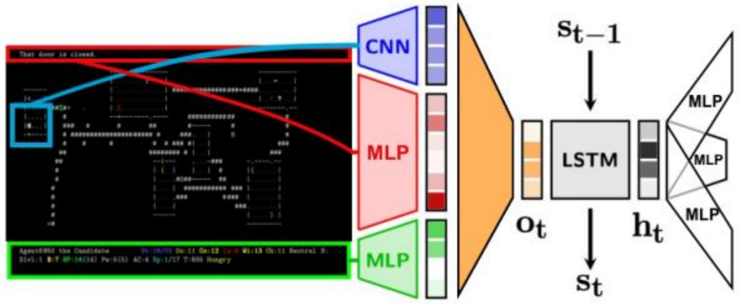
4. Train the complete set of options via RL.
    - Deploy the multi-skill agent in NetHack.
        - When a skill is triggered by policy-over-skills, use the collected sample to train that skill only.
    - Details

→ Better disentangling between skills

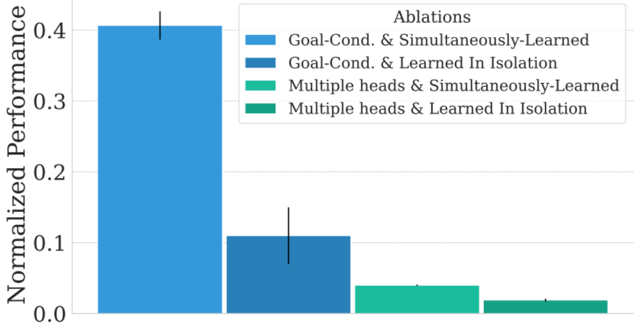
        - CDGPT5 + PPO (asynchronous)
        - Used skill-index (one-hot) conditioned architecture instead of multi-head (gradient interference issue).
        - Trained until every skill has been trained for 2B steps (so I'm guessing at least 10B steps).



(a) Skill-conditioned policy



(b) Multi-head policy



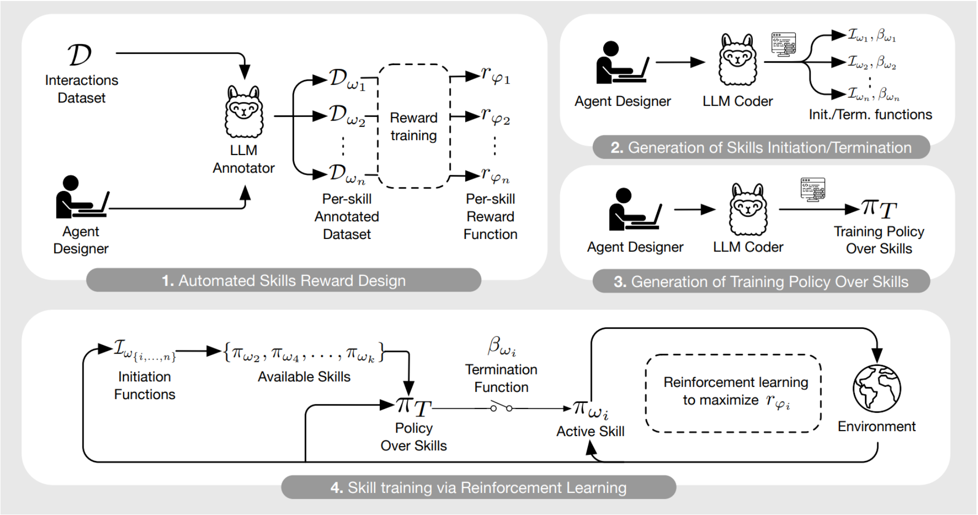
(a) Effect of choices in hierarchical architecture and learning strategy.

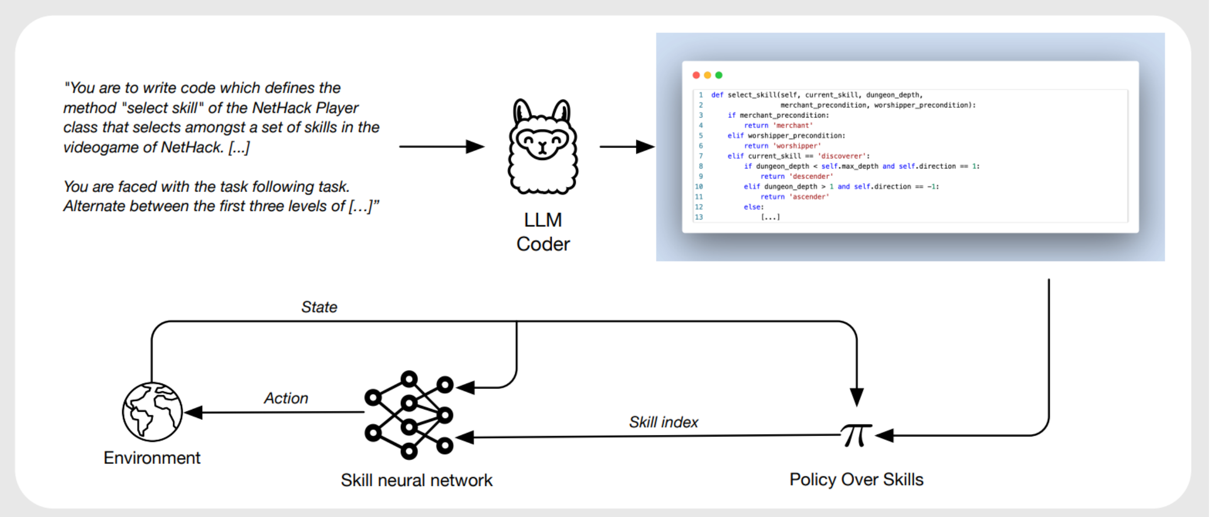
MaestroMotif: LLM-assisted Skill Design

LLM-assisted skill design pipeline

1. Create preference datasets with different prompts asking for distinct behaviors (skills).
2. Make LLM generate codes for when to start/end each skill (I_w, β_w).
3. Make LLM generate code for policy-over-skills (π_T).
4. Train the complete set of options via RL.
5. **For evaluation task, generate another policy-over-skills by explaining the task to LLM!**

→ Zero-shot deploy





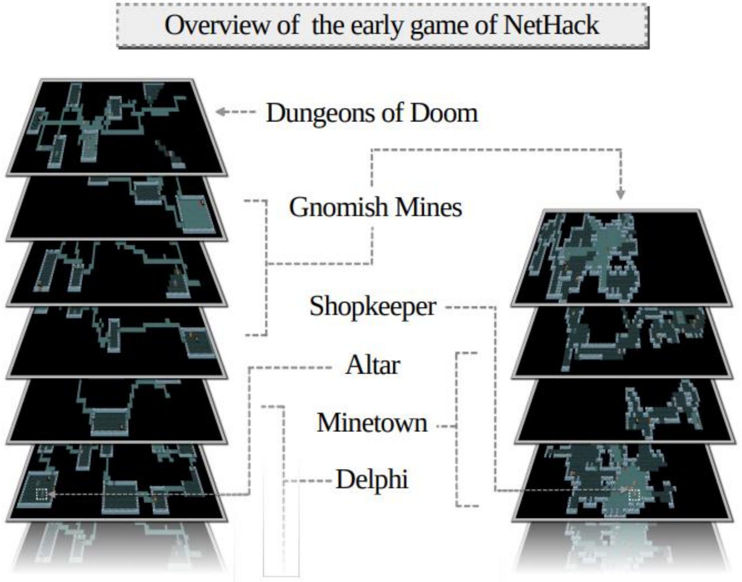
MaestroMotif: LLM-assisted Skill Design

Experimental Setting

- **Navigation tasks**: Reach specific locations (e.g.,Gnomish Mines, Delphi)
- **Interaction tasks**: Interact with certain entities (e.g., Buy/Sell, Identify Blessed/Cursed/Uncursed)
- **Composite tasks**: Sequence of tasks given in language.

Baselines

- Using LLM directly as policy **(LLaMA+ReAct)**
- Applying **Motif** directly on the task
- **Embedding similarity** with pretrained text encoder (e.g., MineClip)
- Giving **score** as priviledged reward signal.
- Multi-skill Motif + LLM-as-policy?
- Multi-skill Motif + task-specific-finetuning + LLM-as-policy?



MaestroMotif: LLM-assisted Skill Design

Quantitative Results

Again, LLMs are bad at handling low-level action space zero-shot

	Zero-shot		Task-specific training		Reward Information
Task	MaestroMotif	LLM Policy	Motif	Emb. Simil.	RL w/ task reward + score
Gnomish Mines	**46% ± 1.70%**	0.1% ± 0.03%	9% ± 2.30%	3% ± 0.10%	3.20% ± 0.27%
Delphi	**29% ± 1.20%**	0% ± 0.00%	2% ± 0.70%	0% ± 0.00%	0.00% ± 0.00%
Minetown	**7.2% ± 0.50%**	0% ± 0.00%	0% ± 0.00%	0% ± 0.00%	0.00% ± 0.00%
Transactions	**0.66 ± 0.01**	0.00 ± 0.00	0.08 ± 0.00	0.00 ± 0.00	0.01% ± 0.00%
Price Identified	**0.47 ± 0.01**	0.00 ± 0.00	0.02 ± 0.00	0.00 ± 0.00	0.00% ± 0.00%
BUC Identified	**1.60 ± 0.01**	0.00 ± 0.00	0.05 ± 0.00	0.00 ± 0.00	0.00% ± 0.00%

Tasks / Methods	**Golden Exit** *"Alternate between the first three levels of the Dungeons of Doom (at least once) until you collect a minimum of 20 gold pieces and defeat 25 monsters; finally try to quit NetHack"*	**Level Up & Sell** *"Do not leave the first dungeon level until you achieve XP level 4, then find a shopkeeper and sell an item that you have collected; finally survive for another 300 steps."*	**Discovery Hunger** *"Reach the oracle level (the Delphi) in the Dungeons of Doom, but not before discovering the Gnomish Mines and eating some food there after getting hungry."*
MaestroMotif	**24.80 % ± 1.18 %**	**7.09% ± 0.99%**	**7.91% ± 1.47%**
LLM Policy	0% ± 0.00%	0% ± 0.00%	0% ± 0.00%
Motif	0% ± 0.00%	0% ± 0.00%	0% ± 0.00%
Embedding Similarity	0% ± 0.00%	0% ± 0.00%	0% ± 0.00%

MaestroMotif: LLM-assisted Skill Design

Quantitative Results

- Comparison to score maximizing algorithms

→ Score may be a rich evaluation signal, but that does not mean that it can carry out complex behaviors!

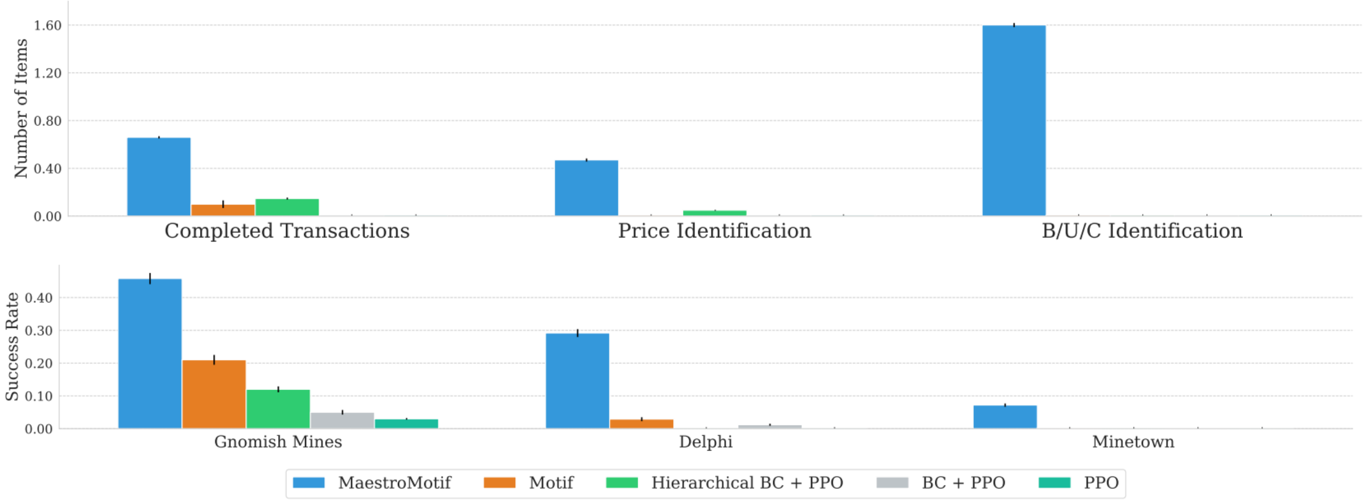


Figure 5: Performance of MaestroMotif and score-maximizing baselines in interaction tasks (first row) and navigation tasks (second row). Despite collecting significant amounts of score, score-maximizing approaches only rarely exhibit any interesting behavior possible in our benchmarking suite.

Summary / Discussion

- Reward/Skill design is labor intensive.
  - **Motif**: High level reward design idea → LLM preference labelling → Train → Aligned RL agent
  - **MaestroMotif**: High level skill descriptions → LLM preference labelling + Coding → Train → Multi-skill agent
- When can we use Motif/MaestroMotif?
  - When dataset for reward labeling (with caption) is available
  - When the cost of training the skill is affordable (e.g., fast, cheap simulation).
  - When the ‘low-level’ is actually trainable (e.g., can’t train humanoid to properly run via pure RL yet).
- Other thoughts
  - LLMs as reward function → VLMs as reward function (e.g., VLM-RM, GenRL)
  - MaestroMotif pipeline can be incorporated into System1-System2 design (specifically System1).
  - MaestroMotif limits its skills to a few. Autonomous skill acquisition (e.g., Voyager)?

Thank You