LLF-Bench: Benchmark for Interactive Learning from Language Feedback

Ching-An Cheng[a*] Andrey Kolobov[a*] Dipendra Misra[a*] Allen Nie[b*] Adith Swaminathan[a*]

Microsoft Research[a] Standford University[b]

Website: https://microsoft.github.io/LLF-Bench

Abstract

We introduce a new benchmark, LLF-Bench (Learning from Language Feedback Benchmark; pronounced as "*elf-bench*"), to evaluate the ability of AI agents to interactively learn from natural language feedback and instructions. Learning from language feedback (LLF) is essential for people, largely because the rich information this feedback provides can help a learner avoid much of trial and error and thereby speed up the learning process. Large Language Models (LLMs) have recently enabled AI agents to comprehend natural language — and hence AI agents can potentially benefit from language feedback during learning like humans do. But existing interactive benchmarks do not assess this crucial capability: they either use numeric reward feedback or require no learning at all (only planning or information retrieval). LLF-Bench is designed to fill this omission. LLF-Bench is a diverse collection of sequential decision-making tasks that includes user recommendation, poem writing, navigation, and robot control. The objective of an agent is to interactively solve these tasks based on their natural-language instructions and the feedback received after taking actions. Crucially, to ensure that the agent actually *learns* from the feedback, LLF-Bench implements several randomization techniques (such as paraphrasing and environment randomization) to ensure that the task isn't familiar to the agent and that the agent is robust to various verbalizations. In addition, LLF-Bench provides a unified OpenAI Gym interface for all its tasks and allows the users to easily configure the information the feedback conveys (among suggestion, explanation, and instantaneous performance) to study how agents respond to different types of feedback. Together, these features make LLF-Bench a unique research platform for developing and testing LLF agents.

1 Introduction

Natural language provides an intuitive medium for a person to teach an AI agent, since that is also how humans learn and teach each other. Compared with rewards, typically used in the reinforcement learning (RL) paradigm (Sutton and Barto, 2018), language feedback can provide rich signals about the agent's behaviors, in addition to a quantitative measure of instantaneous performance. For instance, language feedback can explain why the agent's previous bad behaviors should be avoided, rather than just punishing the agent without giving justification. Language feedback can also provide direct suggestions on how the agent can improve its future behavior, similar to action feedback used in imitation learning (IL) (Ross et al., 2011; Spencer et al., 2021) but easier to provide — after all, it's easier said than done. For example, providing action feedback to a robot requires setting up additional teleoperation devices which might not always be feasible, while language feedback can be given verbally by an ordinary user (Liu et al., 2023a).

LLF-Bench (Learning from Language Feedback Benchmark; pronounced as "*elf-bench*") is a simulation benchmark designed to evaluate an AI agent's ability to *learn* interactively from *just* language

*All authors contributed equally and are alphabetically ordered. Correspondence can be sent to chinganc@microsoft.com, akolobov@microsoft.com, dimisra@microsoft.com, anie@stanford.edu, adswamin@microsoft.com

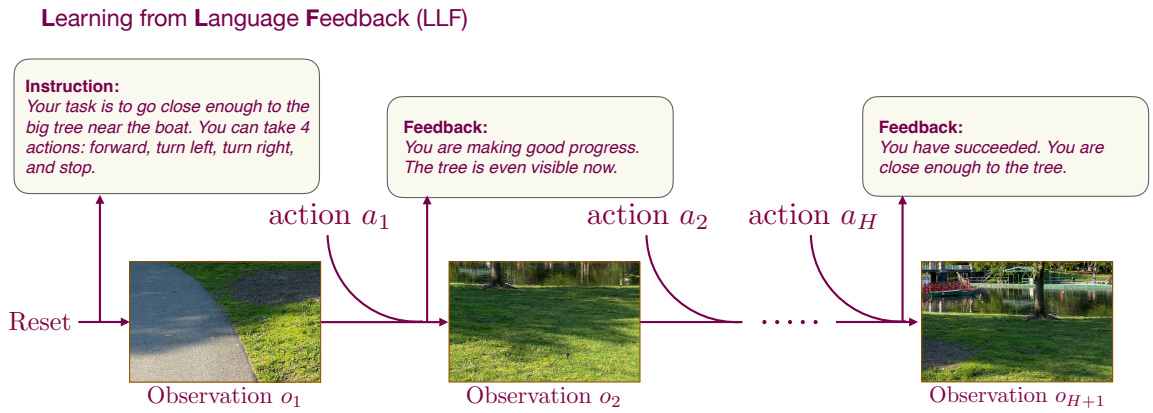

Figure 1: Shows an example navigation task to illustrate our setup, Learning from Language Feedback (LLF). A single episode in LLF starts with a given instruction and can be multi-step long. The actions are taken by the agent that changes the observation and provides a *text feedback* to the agent. The agent receives no reward or any other form of feedback.

feedback. LLF-Bench is a collection of sequential decision making problems (ranging from item recommendation, poem writing, navigation, to robot control). In each problem, an agent interacts with a task environment and receives language instructions and feedback. At the start of an episode, the agent is first given a natural language *instruction* that describes the objective of the task, the rules, and (optionally) side information that may help solve the problem. After executing an action in the environment, the agent receives teacher *feedback* in natural language which can be used as a learning signal.

We call this paradigm *Learning from Language Feedback* (LLF). LLF generalizes reinforcement learning (RL) from reward maximization to general problem solving. Like in RL, LLF focuses on sequential decision problems. However, in contrast to RL, an LLF agent does *not* receive rewards and is *not* necessarily tasked with maximizing returns. Figure 1 shows an example LLF flowchart. LLF replaces RL's assumption of numeric rewards with generic task instructions and feedback expressed in natural language. We can recover RL as an instance of LLF, e.g., with the instruction *"Maximize the accumulated rewards."* and the feedback template *"You've received a reward of X."*. But LLF covers many other scenarios that would be unnecessarily difficult to describe in the conventional RL framing, e.g., training a robotic arm controller by giving it general advice about the types of actions it should consider in certain situations, or asking an agent to write a poem in a certain mood by showing a few examples. We illustrate the similarities and differences between RL and LLF in Figure 3.

How can an interactive agent use language feedback to achieve efficient learning in the LLF setting? A prerequisite is that the agent can already understand commonsense and reason with natural language, so that the agent can focus on skill learning rather than figuring out the basics of language understanding (because instructions, rewards, and feedback in LLF are written in natural language). This has been the main reason why LLF has not received significant attention in the past. However, recently, Large Language Models (e.g., GPT4 (OpenAI, 2023), Gemini (Gemini Team, 2023)) have demonstrated impressive natural language processing abilities. In addition, multiple LLM-agents have shown promising signs of solving text-based problems involving decision making, planning, information retrieval, tool uses (Schick et al., 2023; Wang et al., 2023a; Wu et al., 2023). Therefore, LLMs provide a promising way to work with the general-purpose language feedback in LLF. Further, solving LLF can also be viewed as a way to measure the ability of LLMs to solve new learning tasks. Notably, LLF-Bench differs from the vast majority of benchmarks for evaluating LLMs which are either non-interactive, or allow the agent designer to choose how to verbalize the environment, which can lead to prompt hacking (i.e., an LLM-agent overfitted to a specific environment through its prompts).

1.1 Highlights of LLF-Bench

We design LLF-Bench as a research platform to facilitate the development and testing of LLF agents (e.g., LLM-agents). LLF-Bench consists of 8 diverse sets of decision-making problems (see Figure 2),

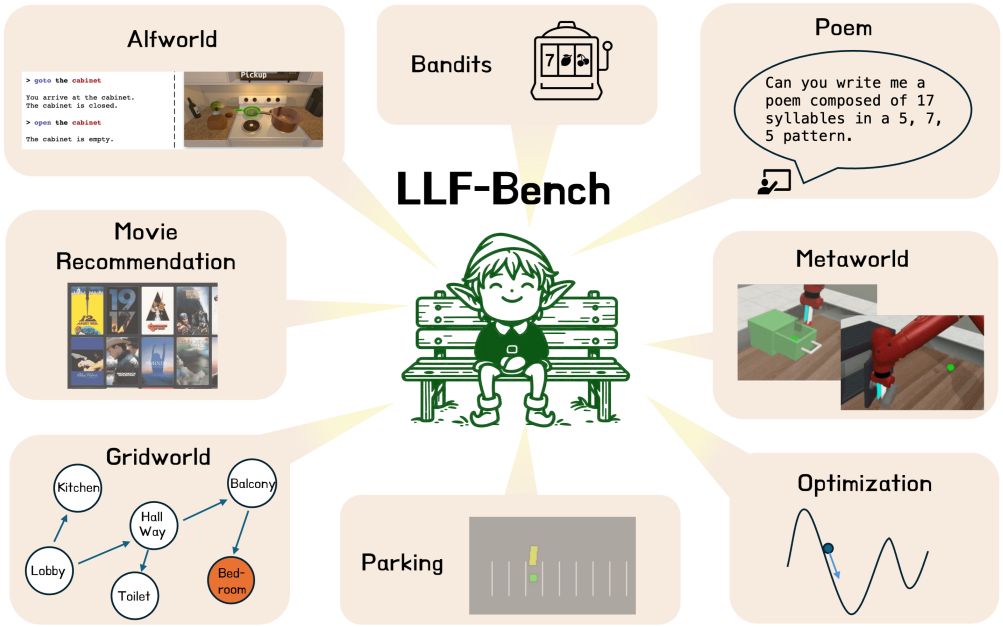

Figure 2: LLF-Bench ("Elf-bench") includes 8 sets of LLF problems. Image by Bing Chat.

with different action spaces (discrete, continuous, and free-form text spaces) and decision horizons:

- `llf-bandit` is a verbalized version of the classic multi-armed bandit problem, which we implement based on gym-bandits. `llf-bandit` tests the agent's learning ability in an unknown environment with a finite number of actions.
- `llf-poem` consists of a set of poem writing tasks, where the agent needs to write a poem satisfying certain syllable- and line-constraints. These problems tests the agent's learning ability to infer and solve constraint satisfication problems.
- `llf-reco-movie` simulates a classic recommendation scenario where a user wants movie or TV show recommendations based on some preferences. The user specifies their preferences in text, and any recommendation made by the agent is matched to a movie database for checking correct preference matching.
- `llf-optimization` consists of 8 loss functions (Rosenbrock, Bohachevsky, etc.) and provides an interface to give verbal feedback for the task of optimization on any loss function.
- `llf-parking` extends the Highway gym environment, providing a long horizon goal-conditioned continuous control task. The agent must control an ego-vehicle to park in a given location without colliding with any obstacles in the environment.
- `llf-gridworld` evaluates the agent's ability to navigate in a graph-based environment. Each node of the graph is a room and the edges are doors connecting the rooms. The agent's goal is to navigate from the room it starts in to the room with treasure.
- `llf-alfworld` adds a wrapper on top of the Alfworld text-based environment (Shridhar et al., 2021) to provide language feedback instead of reward. In `llf-alfworld`, the agent is tasked to solve problems in a text-based house environment. The agent is tested for generalization as each episode can contain a new task in a new house environment.
- `llf-metaworld` is a low-dimensional state-based version of the existing Meta-World v2 benchmark (Yu et al., 2019). It comprises 50 simulated robotic manipulation tasks featuring a Sawyer arm and various objects that this arm needs to bring into desired configurations, such as opening doors, placing cubes in boxes, etc.

When designing a *learning* benchmark, an important consideration is whether the evaluation can truthfully reflect an agent's learning and generalization abilities and separate them from overfitting. To this end, we make two important design choices:

1. Following the framing of LLF, LLF-Bench implements the task instruction as part of the environment, as opposed to as part of the agent. The latter is common in the current literature of LLM-agents, and many LLM-agents heavily rely on using task-specific prompt templates (Wang et al., 2023a; Yao et al., 2023). By this design, we encourage users of LLF-Bench to develop agents that can simultaneously work well across different problems sets in LLF-Bench. We hope that this paradigm shift would facilitate the development of more generic learning agents that can solve multiple tasks, rather than being engineered for solving just a single task.

2. LLF-Bench provides the option to further randomize the textual description of task instruction and feedback that the agent receives. In addition, for several environments, we randomize the environment's latent parameters (e.g., to permute the action ordering in `llf-bandit` or change the room connectivity in `llf-gridworld`) when the environment is reset. Sensitivity to different phrasing of the same instruction is often used to measure the robustness of a text-based model (Ribeiro et al., 2018; Wallace et al., 2019). This design is motivated by the observation that LLMs *as of now*, do not always perfectly understand semantics and can be sensitive to the exact texts that are presented (Zhu et al., 2023). It has been shown LLMs suffer from recency bias and can give drastic different outputs for semantically similar inputs (Arora et al., 2023; Leidinger et al., 2023). To combat that, for each problem instance in LLF-Bench, we manually curate a set of syntax templates via paraphrasing, which are used to produce a diverse yet semantically equivalent set of task instructions and feedback during interactions. By introducing randomization, LLF-Bench can better evaluate the agent's ability of task solving and prevent the agent from overfitting a specific text realization.

One prominent feature of LLF-Bench is its configurable feedback system. Taking inspiration from the education research literature (Shute, 2008), we classify the language feedback into 3 different types: *1)* feedback of the current performance (similar to reward scalars and success booleans), *2)* suggestions of future behaviors (e.g., hints or things to avoid) *3)* explanation of past behavior (e.g., why some behaviors are bad and should not be repeated). By default, a testing environment in LLF-Bench provides a mix of these feedback (when appropriate). It can also be easily configured to provide feedback based on any subset of these categories.

For ease of use, LLF-Bench adopts the OpenAI Gym API (Brockman et al., 2016), which abstracts the interaction with `reset` and `step` API functions. LLF-Bench environments return the natural language instruction and feedback as the observation (a python dict) and the action spaces vary across problems. LLF-Bench environments also return rewards per the Gym `step` API. While agents in the LLF setup do not use rewards, the returned rewards can be used to evaluate an LLF agent's performance; this feature makes the LLF-Bench environments also usable as typical RL environments. LLF-Bench also provides a text-mode option (where both the observation and the action are free-form texts), so that it can also be used as a benchmark for evaluating LLM-agents as well.

1.2 Related Setups and Benchmarks

Many RL environments incorporate natural language. We provide a list summarizing their main features in Table 1. The RL environments can use language to describe the reward/goal (**instructions**), the **observations**, or the **actions**. Commonly, language is used as goal-specifying **instructions** (which is essentially a reward function) for an RL agent (e.g., GridLU by Bahdanau et al. (2019), ViZDoom Text by Chaplot et al. (2018), ISI Block by Misra et al. (2017), and Puddle World by Janner et al. (2018)). In this context, understanding and mapping instructions/goals to the state of the environment is the key challenge. Some RL environments naturally have **observations** in text; these include text-based adventure games (Text World by Côté et al. (2019) and NetHack by Küttler et al. (2020)) and HTML webpages (MiniWoB by Shi et al. (2017), MiniWOB++ by Liu et al. (2018), and WebShop by Yao et al. (2022)). Other RL environments have **action** spaces in text, i.e. an RL agent can generate a sequence of tokens as an action, such as a structured text representing a short executable program (e.g. SHRDLURN by Wang et al. (2016)). However, this was considered challenging due to the relatively large vocabulary space and the difficulty of learning to generate a sequence. None

Environment	Observation Space	Action Space	Reward Space	Language Variations (Robustness)	Language Feedback
	Language Grounding Envs				
SHRDLURN (Wang et al., 2016)	Vector	Text	Scalar	None	No
GridLU (Bahdanau et al., 2019)	Pixel	Discrete	Scalar	None	No
VizDoom Text (Chaplot et al., 2018)	Pixel	Discrete	Scalar	None	No
ISI Block (Misra et al., 2017)	Pixel	Discrete	Scalar	None	No
Puddle World (Janner et al., 2018)	Pixel	Discrete	Scalar	None	No
	Text-based Games				
BabyAI (C-Boisvert et al., 2018)	Pixel	Discrete	Scalar	None	No
Zork (Narasimhan et al., 2015)	Text	Text	Scalar	None	No
TextWorld (Côté et al., 2019)	Text	Text	Scalar	None	No
NetHack (Küttler et al., 2020)	Pixel	Discrete	Scalar	None	No
	Web-Navigation Envs				
MiniWoB (Shi et al., 2017)	Pixel/Text	Disc/Cont	Scalar	None	No
MiniWOB++ (Liu et al., 2018)	Pixel/Text	Disc/Cont	Scalar	Observation	No
WebShop (Yao et al., 2022)	Pixel/Text	Text	Scalar	None	No
	LLM Agent Benchmark Envs				
AgentBench (Liu et al., 2023b)	Text	Text	Scalar	None	No
OpenAGI (Ge et al., 2023)	Text	Text	Scalar	None	No
MINT (Wang et al., 2023b)	Text	Text	Scalar	None	Yes (LLM)
LMRL Gym (Abdulhai et al., 2023)	Text	Text	Scalar	None	No
LLF-Bench (Ours)	Text	All	Scalar[1]+Text	All	Yes (Synthetic)

Table 1: We list several decision-making environments that involve natural language. Language is used to instruct model behavior, represent observation, or is part of the action output. "Language Variations" refers to whether there are multiple descriptions of the same instruction, observation, or reward. "Disc/Cont" means the output is a mix of discrete and continuous variables. **LLF-Bench** offers text representation for instruction, observation, and reward, generates paraphrasing to prevent prompt hacking, and offers procedurally generated synthetic feedback for fast and cheap evaluation.

of these environments provide rewards as text and do not provide feedback on actions. They also do not consider variations in language expressions – such as different phrasing or writing that represent the same underlying goal or state of the environment. Many of these environments are unsuitable for testing LLM agents due to having an observation space that is pixel or vector-based, and the types of tasks are dissimilar to what people use LLMs for today.

On the other hand, several benchmarks have been proposed to evaluate LLM-based agents for decision-making (AgentBench by Liu et al. (2023b), OpenAGI by Ge et al. (2023), MINT by Wang et al. (2023b), and LMRL Gym by Abdulhai et al. (2023)). However, many tasks in these benchmarks center around planning and information retrieval problems. Few require the agent to learn and adapt beyond what an LLM can already do. A real-life user would leverage an LLM-based agent to solve challenging tasks but also give intermediate feedback, such as "make the title text larger" or "wrap the code with an error-catching block." LLF-Bench supports such intermediate feedback as well. Also, due to the lack of language variations, developers might identify a specific prompt that solves an instantiation of the task, over-fitting to a particular writing of the task specification. Lastly, because an LLM-based agent interacts with human users, the specification of reward from a user can often be text. Are LLM-based agents capable of learning and adaptation from rewards represented as text? LLF-Bench aims to provide a set of environments to help answer this question while addressing the challenges in reliably benchmarking LLM-based agents.

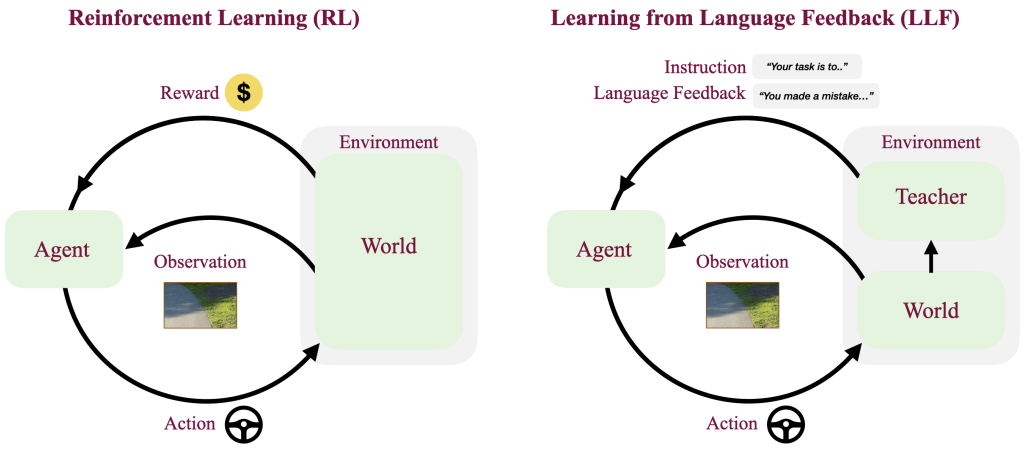

Figure 3: Comparison between RL and LLF setups. LLF replaces reward feedback in RL with language feedback and generalizes the reward maximization objective to general instructions that can be specified via natural language.

2 LLF: Learning from Language Feedback

LLF is a learning paradigm that generalizes the RL setup. As shown in Figure 3, LLF is an abstract setup[2] that models the interaction between an *agent* (e.g., a learning algorithm), a *world* (e.g., a robot hardware, or a recommendation system based on a database), and a *teacher* (e.g., a person who uses or teaches the agent mentioned in Section 1.2).

2.1 Setup

The agent in the LLF is prompted by the teacher to complete a task in the world with some natural language *instruction*. The instruction may be different from reward maximization and could include information about how to interpret observations, what valid actions are, and side information (such as examples) that may help the agent solve the problem. After receiving the instruction, the agent sees the initial observation of the world state, and the agent starts to interact with the world by taking actions within the problem's prescribed action space, which like that in RL can be e.g. a finite space, a continuous vector space, or a free-form text space. After an action is executed, the world's internal state may change and the agent sees the next observation of the world. As the agent interacts and learns to solve the task, the teacher would provide natural language *feedback* to guide the agent to learn better based on how the agent performs. This language feedback is a strict generalization of the reward signal in RL and can provide richer information to help agent learn (e.g., suggestions, explanations, etc.). If we group the world and the teacher in LLF together as an abstract *environment*, we see that LLF mainly replaces the reward maximization objective and feedback in RL with a generic task instruction and language feedback. In LLF-Bench, we simulate LLF problems through the OpenAI Gym interface, which we will describe in Section 3. .

2.2 Motivation

The LLF setup is motivated by the inefficiency and unnaturalness of communicating intentions with rewards. The concept of reward maximization in RL, while giving a simple abstraction of interactive learning, often creates a barrier for humans to transfer knowledge and convey their intention to AI agents. Reward feedback compresses all the information that one wishes to convey down to a single

[1]The scalar reward is for evaluation, not for agent learning in the LLF setup.

[2]Here we follow the convention of Sutton and Barto (2018) that "anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment.". Therefore, we consider the (physical or digital) world that the agent has effects on as well as the teacher who provides instructions and feedback to the agent as part of the environment. As per Sutton and Barto (2018), we note that "the boundary between agent and environment is typically not the same as the physical boundary of a robot's or animal's body. Usually, the boundary is drawn closer to the agent than that."

numerical value, representing signals for encouraging or penalizing certain behaviors. In addition, rewards are received only after the agent takes actions, so the agent would have to not only learn the task solving skill but *also* learn to understand the task's objective. This bottleneck limits the information that can be transferred to the agent and couples skill learning and intention understanding, causing the agent to learn inefficiently in a trial-and-error manner.

Moreover, in many cases, it is difficult for human designers to fully understand the long-term effects of reward maximization, even when each instantaneous reward makes sense. This misalignment has led to many surprising behaviors of RL agents (Amodei et al., 2016). Consequently, reward engineering has been a common practice in building RL systems, where the user iteratively tweaks the reward to give to the agent by observing how the agent behaves after maximizing the current reward function. However, reward engineering is an expensive process. If agents can learn directly from language feedback (i.e. efficiently solving LLF), learning systems can be built more economically.

Overall, compared to RL, LLF encapsulates the rich language feedback that is used in human to human learning. The expressive nature of this rich language feedback, provides a potentially more efficient mechanism for training agents than RL.

2.3 LLF and RL with text/language observations

Interactive learning settings with language-based instructions (Chen et al., 2019; Misra et al., 2018) or observations has been extensively studied (Guu et al., 2017; Wang et al., 2016; Zhong et al., 2021) in the literature. However, in all these settings, one assumes access to either gold actions or rewards that are crucially necessary for understanding the textual instructions and observations, and learning the optimal policy. In contrast, in LLF the agent is neither provided gold actions nor rewards. This makes LLF appear as a harder learning setting than RL. We argue that this difficulty working with general-purpose language feedback, has been the reason why LLF hasn't received much attention previously despite its potential benefits.

However, with the recent success LLMs at general-purpose language reasoning, we have a powerful tool to work with language feedback and develop LLF agents. As mentioned earlier, language feedback can provide more expressive forms of feedback, thereby, providing a more sample-efficient setup for developing agents. For example, the feedback in LLF can be formative which is defined by Shute (2008) as "information communicated to the learner that is intended to *modify* the learner's thinking or behavior for the purpose of *improving* learning.".

We also highlight that with access to accurate LLMs, LLF is not harder than a RL setting, if the task instructions in LLF are detailed enough such that from the observations alone (without language feedback) it is possible for the LLM to infer if the agent has succeeded at following the instruction. Note that this assumption does not mean that the instruction necessarily shows the agent how to solve the problem. Under this assumption, LLF problems can always be solved without the feedback, by a reduction to a RL problem with sparse binary reward of success (the binary reward can be computed using a LLM to detect success based on the instruction and the observation). However, such a reduction approach would lead to inefficient learning. The main research question of LLF is how to best leverage the language feedback, which can convey more information than just success/failure, to learn the optimal policy for the task in a sample-efficient manner. We next describe LLF-Bench as our proposed research platform to measure progress in answering this question.

3 Gym Interface of LLF-Bench

LLF-Bench formalizes a wide variety of decision-making problems by extending the popular OpenAI Gym API. The API contains three key functions — `make`, `reset`, `step` — that are semantically similar to their Gym namesakes and detailed below.

- `make`: Returns an *Environment* object similar to `gym.make`. An LLF-Bench *Environment* extends classic Gym Environments (e.g., with well-defined `ActionSpace` and `ObservationSpace`) with two additional concepts, `instruction` and `feedback`, that are explained below.

- `reset`: After an environment is initialized using `make`, it should be `reset` to receive the initial *Observation* from the *Environment*. LLF-Bench *Observation* is a python dictionary containing `gym.Observation` (i.e., an observation that is contained in the `environment.ObservationSpace`)

as well as `instruction` and `feedback` keys. If the environment uses randomization, then the random number generator can be seeded with the `seed` parameter as input.

- `step`: Takes as input an action that is contained in the `environment.ActionSpace`, and returns a LLF-Bench *Observation* dictionary which includes the `instruction` and `feedback` keys. In addition to the *Observation*, `step` also returns scalar *reward*, boolean flags *truncated* and *terminated* and a miscellaneous *info* dictionary which have the same semantics as Gymnasium environments. An agent for LLF-Bench is expected to solve tasks using the feedback contained within *Observation*, **without** using the *reward* signal. Signals like *reward* and *info* are provided for backward compatibility with Gymnasium and for automated evaluation.

Note that under the hood, LLF-Bench implements all *Environment* objects as compatible with the Gymnasium standard. We provide `EnvironmentCompatibility` wrappers if the *Environment* is instead otherwise compatible with the deprecated Gym (pre-0.21 version) standard. We similarly include `TextWrapper` wrappers that can convert any LLF-Bench *Environment* with bespoke `ObservationSpace` and `ActionSpace` into one with text as the observation and action spaces. This wrapper allows one to directly interface LLM-based agents with LLF-Bench environments and assess their learning and decision-making behavior.

```
1  import llfbench as gym
2
3  # Environments in the benchmark are registered following
4  # the naming convention of verbal-*
5  env = gym.make('verbal-gridworld-v0')
6
7  done = False
8  cumulative_reward = 0.
9
10 # First observation is acquired by resetting the environment
11 observation = env.reset()
12
13 while not done:
14
15     # Observation is dict having 'observation', 'instruction', 'feedback'
16     # Here we print the observation and ask the user for an action
17     action = input( observation['observation'] + '\n' +
18                     observation['instruction'] + '\n' +
19                     observation['feedback'] + '\n' +
20                     'Action: ' )
21
22     # Gridworld has a text action space, so TextWrapper is not needed
23     # to parse a valid action from the input string
24     observation, reward, terminated, truncated, info = env.step(action)
25
26     # reward is never revealed to the agent; only used for evaluation
27     cumulative_reward += reward
28
29     # terminated and truncated follow the same semantics as in Gymnasium
30     done = terminated or truncated
31
32 print(f'Episode reward: {cumulative_reward}')
```

Listing 1: Sample python code snippet for interacting with LLF-Bench environments.

Although each `step` also returns a scalar *reward*, the convention we follow (and recommend to users of LLF-Bench) is that the agent never sees the reward. It can only access the information in *observation*, *instruction* and *feedback* to decide its actions (e.g., see line 17 in Listing 1).

3.1 Instruction and Feedback

Instruction is a string that is defined inside the *Environment* and describes in natural language the problem that a decision-maker must solve. We recommend that agent-designers should not inspect and overfit to a specific instruction describing the desired task in an environment; the default behavior of LLF-Bench environments is to paraphrase instructions in different ways to minimize the chances of prompt overfitting. Three different types of *Instruction* are supported in LLF-Bench, and

can be toggled by passing an appropriate `instruction_type` to the `make` command of a LLF-Bench environment:

- Basic: `instruction_type = 'b'`. This is the default instruction type for LLF-Bench environments. The instructions provide an agent with the goal, semantics of its action space, as well as the expected syntax of its responses. The instruction provides enough information for a competent agent (e.g., a literate human) to begin interacting with the environment.
- Complete: `instruction_type = 'c'`. The instructions additionally provide information to reliably infer (e.g., by a literate human) an optimal policy for achieving the goal.
- Practical: `instruction_type = 'p'`. It contains the *Basic* instructions, and additionally includes *Feedback* for previously executed actions. The goal of a learning agent is to infer the optimal policy (i.e., comparable in performance to the one with `instruction_type = 'c'`) as quickly as possible.

Feedback is a string that provides the signal for an agent to learn from its interaction. LLF-Bench implements two kinds of feedback: an atomic feedback, and a composite feedback. The type of feedback an environment provides to an agent is set by passing an appropriate `feedback_type` parameter to `make`. Atomic feedbacks are inspired by the education research literature (Shute, 2008). LLF-Bench currently supports 5 different types and we plan to include new styles (to include e.g., questioning) in the future:

- `feedback_type = 'r'`: This is the textualization of the scalar reward signal or success signal from classical RL. By using the text-wrapper and this feedback type, several classical RL environments (implemented in OpenAI Gym or Gymnasium) can be comparably tested with LLF agents in LLF-Bench.
- `feedback_type = 'hp'`: This *hindsight positive* feedback provides an explanation about a past action by the agent that was desirable.
- `feedback_type = 'hn'`: This *hindsight negative* feedback provides an explanation about a past action by the agent that was undesirable.
- `feedback_type = 'fp'`: This *future positive* feedback provides a suggestion for a potential future action that could be desirable.
- `feedback_type = 'fn'`: This *future negative* feedback provides a suggestion for potential future actions that should be avoided.

`feedback_type = 'r'` corresponds to the **current performance** evaluation from the education research literature, whereas `feedback_type = 'fp', 'fn'` correspond to **future behavior** suggestion. Finally, `feedback_type = 'hp', 'hn'` correspond to the **past behavior** explanation style of feedback studied in the education research literature.

Composite feedback types allow the environment to provide the agent multiple kinds of atomic feedbacks. This makes for a more realistic learning problem, rather than the same type of atomic feedback at every `step` of the environment.

- `feedback_type = 'a'`: *All* of the Atomic feedback types that are supported by the environment are provided to the agent at each round of interaction.
- `feedback_type = 'm'`: The agent receives a *Mix* of different atomic feedbacks. A random subset of the supported feedback types are sampled by LLF-Bench to provide to the agent at each step.
- `feedback_type = 'n'`: The agent receives *No* feedback, this mode is provided for debugging purposes.

The `make` API accepts any of the composite feedback types, or any subset of the atomic feedback types to allow fine-grained control of the learning signal that an agent can receive from LLF-Bench environments. The default behavior in `make` for any environment uses `feedback_type = 'a'`. See Listing 1 for sample code that creates a LLF-Bench environment and instantiates an agent to interact with it using `make`, `reset` and `step` API calls.

Problem Set	Action Space	Horizon	Stateful	Instruction	Feedback
`llf-bandit`	Discrete	1	No	b, p, c	all
`llf-poem`	Text	1	No	b	all
`llf-reco-movie`	Text	1	No	b	all
`llf-optimization`	Continuous	10	Yes	b	all
`llf-parking`	Continuous	100	Yes	b	r, hp, hn
`llf-gridworld`	Finite	20	Yes	b, p, c	all
`llf-alfworld`	Text	100	Yes	b	all
`llf-metaworld`	Continuous	30	Yes	b	r, hp, hn, fp

Table 2: Properties of problem sets included in LLF-Bench. Instruction and Feedback column denote the types of instruction and feedback that are supported by the environment. If feedback is all, then it means that all 5 feedback (r, hn, hp, fn, and fp) are supported.

3.2 Instruction and Feedback Randomization

To reduce the sensitivity of learning agents to a specific text realization, LLF-Bench implements a template-based paraphrasing system, by which users can randomize the instruction and the feedback that the agent receives. For each problem in LLF-Bench, we implement about 4-20 paraphrased templates for each instruction and each feedback type. When the randomization options are turned on, the LLF-Bench environment will randomly choose one from these curated templates to formulate the language instruction and feedback returned to the agent. LLF-Bench also provides the option to deterministically use a particular template. The randomness of paraphrasing can be controlled by setting the `seed` parameter in the OpenAI Gym `reset` function.

4 Tasks in LLF-Bench

LLF-Bench consists of 8 different problem sets, ranging from user-recommendation, poem-writing, navigation, to robot control. In the LLF setup, the reward is masked out (though the environments in LLF-Bench still return rewards for evaluation purposes). To solve these problem efficiently, an LLF agent needs to have sufficient common sense understanding of the natural language instruction and the feedback. In addition, the agent needs to be able to *learn* from environmental interactions and feedback. We intentionally design these suites of problems such that, while the agent can tell success from the instruction and the environmental observation, it is difficult for the agent to infer the optimal policy from them without additional learning.

These problem sets feature different action spaces, problem horizons, and test different abilities of LLF agents. We provide a summary in Table 2 and next describe each problem set in more detail.

4.1 `llf-bandit`

`llf-bandit` is a verbalized version of the classic multi-armed bandit problem. We built `llf-bandit` based on gym-bandits by adding natural language task instruction and feedback. There are a total of 8 bandit problems in `llf-bandit`. For each problem, the task instruction tells the task name from the underlying `gym-bandits`, that the goal is a bandit problem, as well as the feasible actions. While being a bandit problem, `llf-bandit`'s feedback does not necessarily convey the reward value in text (it depends on the configuration of the feedback type). When `reset`, the environment randomizes the order of actions and, if applicable, the underlying reward function. The agent here needs to learn to explore and exploit in multiple rounds of interactions to find the best arm as fast as possible with small regret (measured in terms of the hidden rewards). Overall, `llf-bandit` tests the agent's learning ability in an unknown environment with a finite number of actions.

4.2 `llf-poem`

`llf-poem` is a collection of text-generation tasks requiring a poem to be written that conforms to a particular number of lines and number of syllables for each line. Even though there are many types of formal poems, the current set of tasks supports basic types that follow syllable and line-based

constraints. Such formal poems include Haiku (a three-line short poem following a 5-7-5 syllable pattern), Tanka (a five-line short poem following a 5-7-5-7-7 pattern), and custom environments where a user can specify the number of lines and how many syllables per line. We use the CMU Pronouncing Dictionary for syllable verification[3]. `llf-poem` provides detailed fine-grained feedback on each line – a good environment to test whether the LLM-based agents can improve quickly given feedback.

4.3 `llf-reco-movie`

`llf-reco-movie` is an environment that simulates user-recommendation system interactions on the topic of recommending movies. To simulate a user, the environment will first randomly sample a user preference profile over a set of attributes such as the type of entertainment (TV show or movie), year range (80s, 90s, 2000s, or recent), preferred genres (Action, Comedy, Documentary, etc.), and age restriction (child/family-friendly or R-rated). An agent needs to recommend a few items (no restriction on the number of items) that all satisfy the stated preference. An item-by-item feedback is provided in this environment to point out detailed preference violations that can allow LLMs to improve their recommendations.

4.4 `llf-optimization`

`llf-optimization` provides an easy-to-use interface with automatic procedurally generated feedback that examines LLMs' ability to make a series of proposals x to minimize a particular loss function $y = f(x)$. The feedback provided in this environment is created by computing gradient $\frac{dy}{dx}$ and then verbalizing this information based on the change in input between the previously proposed x and the current chosen x. We provide implementations of 8 classic loss functions (Rosenbrock, Bohachevsky, etc.), and the base class is easily extendable to cover other loss functions. This is an environment where we can measure LLM's ability to make decisions with observed information on an unknown loss landscape.

4.5 `llf-parking`

`llf-parking` extends the Highway gym environment to LLF-Bench. It is a long horizon goal-conditioned continuous control task where the agent can manipulate the throttle and steering input to an ego-vehicle. It must park the ego-vehicle in a given location without colliding with any obstacles in the environment. We extended the environment by (1) describing the observation and action spaces in text, and (2) verbalizing the per-time-step reward (distance to goal) to provide text feedback about goal progress and obstacle collisions. An agent must learn how its control inputs affect the vehicle's dynamics, and plan to accomplish the eventual parking goal.

4.6 `llf-gridworld`

The `llf-gridworld` domain models a navigation agent in a graph-based gridworld. The world is represented by a graph where rooms are denoted by nodes and edges denote doors. A room can have at most 4 doors along the north, south, east and west direction. These directions form the agent's action space. At any given time, the agent is in exactly one of the rooms. If the agent takes an action, such as $a = \textit{north}$, then it will transition from its current room, to the room connected by the door along the north direction, if one exists. If no such door exists, then the agent stays in the same room. All transitions are deterministic. A room can contain many different types of objects. A unique room, called the treasure room, contains the treasure object. The agent starts in a start room and its goal is to navigate to the treasure room. The number of rooms, objects, object type, and distance to the treasure, can be easily customized.

The agent's observation describes the current room including all the objects in it, and the different doors that are available. The agent can get all 5 types of feedback: `r`, `hn`, `hp`, `fn`, and `fp`. For example, for the `fn` feedback, the agent will be told which action, i.e., a door, to avoid going through in the next step.

[3] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

4.7 `llf-alfworld`

The `llf-alfworld` environment is a wrapper built on top of the popular `AlfWorld` text-game environment (Shridhar et al., 2021) which itself is built as a parallel to the embodied `Alfred` dataset (Shridhar et al., 2020). `llf-alfworld` contains multi-step reasoning tasks, where in each episode, the agent is given an instruction in a house setting and must take a sequence of actions to fulfill this instruction. In each step, the agent is given a textual description of what it sees and a list of valid actions. The agent generates a text action (e.g., *open drawer 1*), which if it is valid can change the agent's observation, and if it is invalid then results in no change. The agent additionally gets a reward for each action. The goal of the agent is to maximize the total reward by solving the task. Unlike the `llf-gridworld` setting, the agent is tested for generalization as each episode can contain a new task in a possibly new house environment.

The main addition in `llf-alfworld` is the capability to provide text feedback instead of reward. The text feedback is generated using an optimal trajectory for that episode, as well as the instantaneous reward and the list of valid commands for each time step. Similar to `llf-gridworld`, the agent can get all 5 types of feedback: `r`, `hn`, `hp`, `fn`, and `fp`.

4.8 `llf-metaworld`

`llf-metaworld` wraps the low-dimensional fully observable state-based version of the existing Meta-World v2 benchmark (Yu et al., 2019) into a textual interface. Meta-World consists of 50 simulated robotic manipulation tasks, in each of which a robotic Sawyer arm needs to move an object into a specified position, e.g., push a puck to a goal location or press a button. An agent trying to accomplish an `llf-metaworld` task is presented with an instruction stating that the task is about getting a robotic manipulator to successfully handle an object and explaining what each dimension of the agent's 4D state space means. By default, the environment interprets an agent's action as a target pose where the arm should move[4], and tries to move the arm there using Meta-World's built-in P-controller. At each time step, the agent receives as observation a description of the current state, mentioning the pose of the arm, and all relevant objects in the scene. The language feedback here may include advice on where to move the arm next and where not to move it.

5 Related Work

Grounded Language Learning Reinforcement learning with textual information has been studied under the branch of multi-modal representation learning. This branch of study has several focuses that are both similar and different from our goal with LLF-Bench. One focus deals with ambiguity and difficulty in understanding instructions or goals specified by natural language (Bahdanau et al., 2019; Chaplot et al., 2018; Wang et al., 2016). While the ambiguity of instructions is a concern, we focus more on robustly behaving under different instructions that all represent the same underlying goal. Another focus of this body of work is to ground visual information with textual instruction – a core aim of multi-modal representation learning (Bisk et al., 2016; Misra et al., 2017), with an extension to robotic interaction (Karamcheti et al., 2022, 2023). Language provides a natural shared representation that enables easier transfer between different tasks (Hanjie et al., 2021) or supplies important information such as safety constraints for a policy (Yang et al., 2021). In previous work, feedback is often not considered. When feedback is considered, it is usually framed as error messages from a syntax parser (if the action space is text) and can indeed be incorporated into learning (Côté et al., 2019). This type of feedback corresponds to `feedback_type = 'hn'` in our setup.

Text-based Games Extending from using reinforcement learning for solving complex games, there are many text-based games that include challenges such as the navigation of space, manipulation of the environment to achieve goals, and reaction to random events. Narasimhan et al. (2015) repurposed a classic text adventure game, Zork, where both observation and action space are text. Côté et al. (2019)

[4]The dynamics of `llf-metaworld` differs from the one in the original Meta-World. Here the agent controls the target location (the simulator runs the P-controller to act in the original Meta-World environment for several steps until the target location is reached or it is timed out), whereas in the original environment the agent controls force to incrementally change the end-effector. This design is to make the problem horizon shorter and more closely mimic the common use cases of industrial robotic manipulators.

proposed a set of text-based game environments and included a few carefully designed challenges for RL to solve, such as large state and action space (determined by the vocabulary size) and long credit assignment. On the other spectrum, Küttler et al. (2020) created a learning environment from the game NetHack. Although the game state is represented with hundreds of text symbols, policy learning is conducted on the screenshot of the terminal. Similarly, BabyAI (Chevalier-Boisvert et al., 2019) is a set of procedurally generated grid-like maze environments – the objects and representation in the environment are a fixed set of symbols. None of these environments consider providing language feedback on the agent's action.

Learning from Language Feedback Providing feedback to an RL agent's action as part of the learning signal beyond task rewards has been studied in robotics. However, most of the efforts were limited to eliciting binary preference feedback (Biyik and Sadigh, 2018; Sadigh et al., 2017) or ranking-based feedback from real people (Basu et al., 2019). Sumers et al. (2021) crowd-sourced a small feedback dataset on a small game. They considered three types of feedback, evaluative feedback (which corresponds to `feedback_type = 'r'`), descriptive feedback (which in our setup is decomposed into `feedback_type = 'hp', 'hn'`), and imperative feedback (which corresponds to `feedback_type = 'fp', 'fn'`). They then used a sentiment classifier to extract coarse information from this feedback to improve the policy's behavior. Nguyen et al. (2021) proposed an approach to map textual instructions to trajectories in embodied settings by assuming that a user can label a generated trajectory with the instruction that is likely to generate the trajectory under the optimal policy. More recently, Cui et al. (2023); Liu et al. (2023a) studied the case of language feedback as corrections to a robotic arm at any time of the task execution, which is an instance of the LLF setup that we are considering.

LLM Sensitivity to Prompts A long line of work has investigated smaller-scale language-based systems' sensitivity to different expressions that have the same underlying meaning. They can be categorized as adversarial attacks to text-based systems (Ribeiro et al., 2018; Wallace et al., 2019) or as mechanisms to improve language-based systems' output via self-consistency (Edunov et al., 2018). More recently, the lack of robustness to prompts has been found on large language models as well (Liu et al., 2023c; Wolf et al., 2023). Zhu et al. (2023) proposed a benchmark dataset to investigate the robustness of LLMs on different types of prompts that can contain user errors for tasks related to natural language.

LLM Agent Benchmarks Unlike previous efforts to incorporate language to develop RL policies, building agents using LLM has ushered in a new set of challenges. In general, the environments included in these benchmarks only concern planning and information retrieval with sparse reward signals at the end. Very few of these benchmarks measure the ability of an agent to learn and adapt to a task (e.g., the Abstraction and Reasoning Corpus by Chollet (2019)). Liu et al. (2023b) proposed a set of environments that cover a few popular types of task setups, such as web browsing, game, and code generation. Their focus is on the diversity of tasks, not the robustness of LLMs or how well they can incorporate feedback, which is dissimilar to how LLMs are currently being used in a user-centered environment. Ge et al. (2023) constructed a set of tasks where LLMs are prompted to use language or vision-related models to solve a complex task that requires multiple steps. The task-level feedback they provide is a numerical score from a domain-specific evaluation method. MINT (Wang et al., 2023b) is a benchmark that also offers natural language style feedback. However, MINT synthesizes user feedback by prompting LLMs. This incurs additional costs, introduces additional variability in the evaluation process, and makes it challenging to represent the diversity of human feedback styles. LMRL Gym (Abdulhai et al., 2023) provides a set of 8 environments that include full and partial observability. The tasks are similar to language-grounding tasks and text games. However, no interim feedback is provided during multi-round interactions.

Acknowledgement

We are thankful to Marc-Alexandre Cote and Victor Zhong for helping us understand the Alfworld environment. We thank Christine Herlihy for helpful discussions on the Movie Recommendation environment. We gratefully acknowledge Ahmed Awadallah and John Langford for their organizational

support. Part of this work was done when Allen Nie was an intern at Microsoft Research. We also appreciate ChatGPT and GPT4 for providing some of the paraphrases used in LLF-Bench and BingChat for generating the elf-bench image.

References

M. Abdulhai, I. White, C. Snell, C. Sun, J. Hong, Y. Zhai, K. Xu, and S. Levine. LMRL Gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv preprint arXiv:2311.18232*, 2023.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

S. Arora, A. Narayan, M. F. Chen, L. Orr, N. Guha, K. Bhatia, I. Chami, and C. Re. Ask me anything: A simple strategy for prompting language models. In *ICLR*, 2023.

D. Bahdanau, F. Hill, J. Leike, E. Hughes, P. Kohli, and E. Grefenstette. Learning to understand goal specifications by modelling reward. In *ICLR*, 2019.

C. Basu, E. Bıyık, Z. He, M. Singhal, and D. Sadigh. Active learning of reward dynamics from hierarchical queries. In *IROS*, 2019.

Y. Bisk, D. Yuret, and D. Marcu. Natural language communication with robots. In *NAACL-HLT*, 2016.

E. Biyik and D. Sadigh. Batch active preference-based learning of reward functions. In *CORL*, 2018.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2018.

H. Chen, A. Suhr, D. Misra, N. Snavely, and Y. Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *CVPR*, 2019.

M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *ICLR*, 2019.

F. Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

M.-A. Côté, A. Kádár, X. Yuan, B. Kybartas, T. Barnes, E. Fine, J. Moore, M. Hausknecht, L. El Asri, M. Adada, W. Tay, and A. Trischler. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018 Held in Conjunction with IJCAI 2018*. Springer, 2019.

Y. Cui, S. Karamcheti, R. Palleti, N. Shivakumar, P. Liang, and D. Sadigh. No, to the right: Online language corrections for robotic manipulation via shared autonomy. In *HRI*, 2023.

S. Edunov, M. Ott, M. Auli, and D. Grangier. Understanding back-translation at scale. In *EMNLP*, 2018.

Y. Ge, W. Hua, J. Ji, J. Tan, S. Xu, and Y. Zhang. OpenAGI: When LLM meets domain experts. *arXiv preprint arXiv:2304.04370*, 2023.

Gemini Team. Gemini: A family of highly capable multimodal models. https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf, 2023. Accessed: December-10-2023.

K. Guu, P. Pasupat, E. Liu, and P. Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *ACL*, 2017.

A. W. Hanjie, V. Y. Zhong, and K. Narasimhan. Grounding language to entities and dynamics for generalization in reinforcement learning. In *ICML*, 2021.

M. Janner, K. Narasimhan, and R. Barzilay. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6:49–61, 2018.

S. Karamcheti, M. Srivastava, P. Liang, and D. Sadigh. Lila: Language-informed latent actions. In *CORL*, 2022.

S. Karamcheti, S. Nair, A. S. Chen, T. Kollar, C. Finn, D. Sadigh, and P. Liang. Language-driven representation learning for robotics. *arXiv preprint arXiv:2302.12766*, 2023.

H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The NetHack learning environment. In *NeurIPS*, 2020.

A. Leidinger, R. van Rooij, and E. Shutova. The language of prompting: What linguistic properties make a prompt successful? *arXiv preprint arXiv:2311.01967*, 2023.

E. Z. Liu, K. Guu, P. Pasupat, and P. Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *ICLR*, 2018.

H. Liu, A. Chen, Y. Zhu, A. Swaminathan, A. Kolobov, and C.-A. Cheng. Interactive robot learning from verbal correction. *arXiv preprint arXiv:2310.17555*, 2023a.

X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688*, 2023b.

Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, and Y. Liu. Jailbreaking ChatGPT via prompt engineering: An empirical study. *arXiv preprint arXiv:2305.13860*, 2023c.

D. Misra, J. Langford, and Y. Artzi. Mapping instructions and visual observations to actions with reinforcement learning. In *EMNLP*, 2017.

D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi. Mapping instructions to actions in 3D environments with visual goal prediction. In *EMNLP*, 2018.

K. Narasimhan, T. D. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. In *EMNLP*, 2015.

K. X. Nguyen, D. Misra, R. Schapire, M. Dudík, and P. Shafto. Interactive learning from activity description. In *ICML*, 2021.

OpenAI. GPT-4 technical report, 2023.

M. T. Ribeiro, S. Singh, and C. Guestrin. Semantically equivalent adversarial rules for debugging nlp models. In *ACL*, 2018.

S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011.

D. Sadigh, A. D. Dragan, S. Sastry, and S. A. Seshia. Active preference-based learning of reward functions. In *RSS*, 2017.

T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang. World of Bits: An open-domain platform for web-based agents. In *ICML*, 2017.

M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *CVPR*, 2020.

M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*, 2021.

V. J. Shute. Focus on formative feedback. *Review of educational research*, 78:153–189, 2008.

J. Spencer, S. Choudhury, A. Venkatraman, B. Ziebart, and J. A. Bagnell. Feedback in imitation learning: The three regimes of covariate shift. *arXiv preprint arXiv:2102.02872*, 2021.

T. R. Sumers, M. K. Ho, R. D. Hawkins, K. Narasimhan, and T. L. Griffiths. Learning rewards from linguistic feedback. In *AAAI*, 2021.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh. Universal adversarial triggers for attacking and analyzing NLP. In *EMNLP*, 2019.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

S. I. Wang, P. Liang, and C. D. Manning. Learning language games through interaction. In *ACL*, 2016.

X. Wang, Z. Wang, J. Liu, Y. Chen, L. Yuan, H. Peng, and H. Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*, 2023b.

Y. Wolf, N. Wies, Y. Levine, and A. Shashua. Fundamental limitations of alignment in large language models. *arXiv preprint arXiv:2304.11082*, 2023.

Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

T.-Y. Yang, M. Hu, Y. Chow, P. Ramadge, and K. R. Narasimhan. Safe reinforcement learning with natural language constraints. In *NeurIPS*, 2021.

S. Yao, H. Chen, J. Yang, and K. R. Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

T. Yu, D. Quillen, Z. He, R. Julian, A. Narayan, H. Shively, A. Bellathur, K. Hausman, C. Finn, and S. Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CoRL*, 2019.

V. Zhong, H. A. Wang, S. Wang, K. R. Narasimhan, and L. Zettlemoyer. SILG: The multi-domain symbolic interactive language grounding benchmark. In *NeurIPS*, 2021.

K. Zhu, J. Wang, J. Zhou, Z. Wang, H. Chen, Y. Wang, L. Yang, W. Ye, Y. Zhang, N. Z. Gong, and X. Xie. PromptBench: Towards evaluating the robustness of large language models on adversarial prompts. *arXiv preprint arXiv:2306.04528*, 2023.