MLGym: A New Framework and Benchmark for Advancing AI Research Agents

Deepak Nathani[1♠◇], **Lovish Madaan**[2,7], **Nicholas Roberts**[3◇], **Nikolay Bashlykov**[7]
Ajay Menon[7], **Vincent Moens**[6], **Mikhail Plekhanov**[7], **Amar Budhiraja**[7]
Despoina Magka[5], **Valdislav Vorotilov**[7], **Gaurav Chaurasia**[7], **Dieuwke Hupkes**[7]
Ricardo Silveira Cabral[7], **Tatiana Shavrina**[7], **Jakob Nicolaus Foerster**[4,5]
Yoram Bachrach[5], **William Yang Wang**[1], **Roberta Railenau**[7♠]

[1] University of California, Santa Barbara [2] University College London
[3] University of Wisconsin-Madison [4] University of Oxford [5] FAIR at Meta
[6] PyTorch Core Libraries at Meta [7] GenAI at Meta

Abstract

We introduce MLGym and MLGym-Bench, a new framework and benchmark for evaluating and developing LLM agents on AI research tasks. This is the first Gym environment for machine learning (ML) tasks, enabling research on reinforcement learning (RL) algorithms for training such agents. MLGym-bench consists of 13 diverse and open-ended AI research tasks from diverse domains such as computer vision, natural language processing, reinforcement learning, and game theory. Solving these tasks requires real-world AI research skills such as generating new ideas and hypotheses, creating and processing data, implementing ML methods, training models, running experiments, analyzing the results, and iterating through this process to improve on a given task. MLGym makes it easy to add new tasks, integrate and evaluate models or agents, generate synthetic data at scale, as well as develop new learning algorithms for training agents on AI research tasks. We evaluate a number of frontier large language models (LLMs) on MLGym-Bench and observe that they can improve on the given baselines, usually by finding better hyperparameters, but do not generate novel hypotheses, algorithms, architectures, or substantial improvements. We open-source our source code[1] for MLGym framework, and benchmark to facilitate future research in advancing the AI research capabilities of LLM agents.

1 Introduction

Accelerating scientific discovery has been a long-standing ambition in artificial intelligence (AI) research, with early initiatives like the Oak Ridge Applied Artificial Intelligence Project in 1979 (Team, 1985; Emrich et al., 1988; Johnson & Schaffer, 1994). More recent explorations enabled by advances in foundation models (Achiam et al., 2023; Anthropic, 2024; Team et al., 2024; Dubey et al., 2024) provide a proof-of-concept of a fully automated pipeline for end-to-end paper generation (Lu et al., 2024). In the future, we envision AI Research Agents capable of independently conducting literature search, generating scientific hypotheses, designing experiments, implementing new methods, analyzing results, disseminating findings by writing scientific papers, and applying this research in products, thus assisting with all parts of the research process. Unlike traditional methods, LLM agents could reveal hidden interdisciplinary relationships given their vast cross-domain knowledge, leading to novel insights and solutions to open research problems. Machine learning (ML) research, with its

[1] https://github.com/facebookresearch/MLGym
◇ Work done during internship at GenAI at Meta.
♠ Corresponding Authors. Email at dnathani@ucsb.edu or raileanu.roberta@gmail.com

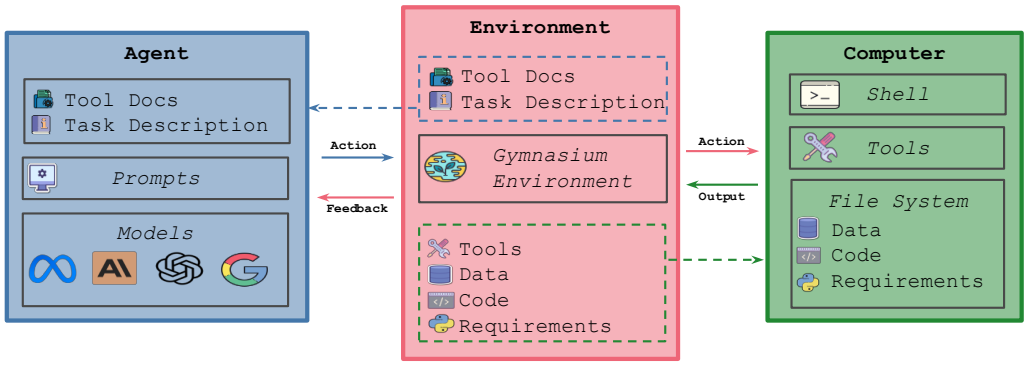

Figure 1: Diagram of MLGym, a unified framework for developing and evaluating LLM agents on diverse open-ended AI research tasks.

emphasis on empirical validation and systematic experimentation in simulation, presents an ideal testbed for exploring and improving the utlity of LLMs for advancing scientific research.

However, the scientific method inherently relies on empirical validation, rigorous evaluation, and standardized benchmarks to ensure the reliability and reproducibility of findings. While significant progress has been made in developing AI agents for various domains (Yang et al., 2024; Wu et al., 2024; Ma et al., 2024; Deng et al., 2023; Wang et al., 2023), we currently lack comprehensive frameworks and benchmarks specifically designed to assess their capabilities in conducting open-ended AI research in diverse domains. This absence of standardized evaluation tools hinders our ability to objectively measure progress and identify areas for improvement in this emerging field.

Recently, a number of papers have started to evaluate LLM agents on various SWE and ML tasks; notable examples include SWE-Bench (Jimenez et al., 2023), SWE-agent (Yang et al., 2024), ScienceAgentBench (Chen et al., 2024), SUPER (Bogin et al., 2024), MLE-Bench (Chan et al., 2024), MLAgentBench (Huang et al., 2024), and RE-Bench (METR, 2024). However, existing benchmarks for AI Research Agents either do not include open-ended research tasks, or only cover a narrow range of research domains. In addition, existing frameworks are not designed to enable research on different training algorithms for AI Research Agents such as reinforcement learning, curriculum learning, or open-ended learning. Finally, current frameworks do not allow flexible artifacts to be evaluated (e.g. different outputs of the agent's research such as a model, algorithm, or set of predictions).

In this paper, we introduce MLGym—the first Gym (Brockman et al., 2016) environment for AI Research Agents and a unified framework designed to integrate diverse and open-ended AI research tasks into a single platform for developing and evaluating LLM agents on such tasks (see Figure 1 for a diagram of MLGym). Being a Gym environment, our framework enables research on different training algorithms for AI Research Agents such as reinforcement learning (RL), curriculum learning, and open-ended learning. We also release MLGym-Bench, a curated set of 13 open-ended research tasks across computer vision, natural language processing, reinforcement learning, and game theory.

MLGym flexibly evaluates various artifacts including CSV outputs, model weights, RL algorithms, or game theory strategy code. We compare a number of frontier LLMs on MLGym-Bench and propose an evaluation metric adapted from the optimization (Dolan & Moré, 2002) and AutoML (Roberts et al., 2022a) literature to assess relative performance across tasks. Finally, we introduce a hierarchical framework to categorize the capabilities of AI agents for accelerating AI research (see section 3).

Benchmark	Gym Interface	Algorithmic Tasks	Open-Ended Research	Flexible Artifacts	Agentic Harness
MLGym (ours)	✓	✓	✓	✓	✓
MLE-Bench	✗	✗	✗	✗	✗
SWE-Bench/Agent	✗	✗	✗	✗	✓
MLAgentBench	✗	✗	✓	✓	✓
RE-Bench	✗	✗	✓	✓	✗
ScienceAgentBench	✗	✗	✗	✗	✗

Table 1: Comparison of MLGym with other related LLM agent frameworks and benchmarks. Algorithmic Tasks refers to tasks that require coming up with new algorithms such as game theory or SAT problems. Open-ended Research refers to the tasks that are not fully solved by the research community and where multiple new solutions could be discovered such as language modeling, game theory or SAT problems. Flexible Artifacts refers to the ability to evaluate different research artifacts such as model weights, reinforcement learning algorithms, or code capturing an agent's strategy. Agentic Harness corresponds to benchmarks that provide a default harness for evaluating agents.

2 Related Work

Table 1 shows a comparison between MLGym and MLGym-Bench with other related LLM agent frameworks and benchmarks.

First, MLGym is the first framework for AI Research Agents that provides a Gym interface, making it easy to integrate and train these agents using RL algoritms. MLGym-Bench is also the first benchmark to include tasks that require research on algorithms in multiple domains such as RL, game theory, or SAT.

Second, MLGym-Bench encompasses a wide range of open-ended AI research tasks, covering supervised learning, language modeling, reinforcement learning, game theory and SAT. In contrast, SWE-Bench/SWE-Agent (Yang et al., 2024) focuses on solving Github issues so the code changes either fix the code or not (as opposed to optmization tasks with finer-grained metrics, such as a loss metric in a supervised learning problem). Similarly, MLE-Bench (Chan et al., 2024) includes narrowly scoped machine learning tasks from Kaggle competitions. While these tasks have a spectrum of quality levels, they tend to be already solved by current state-of-the-art methods. On the other hand, MLAgentBench (Huang et al., 2024) contains both ML-specialized tasks (regression, classification, code speed improvements) and tasks focused on recent research challenges (e.g. CLRS reasoning corpus (Veličković et al., 2022), BabyLM challenge (Warstadt et al., 2023)). RE-bench (METR, 2024) also consists of broadly scoped ML engineering tasks which are hard to saturate and reward increasingly sophisticated approaches. ScienceAgentBench (Chen et al., 2024) incorporates data-driven scientific discovery tasks extracted from peer-reviewed publications, but which are so specific that they resemble Kaggle competition rather than open research questions.

Third, MLGym allows for flexible evaluation artifacts: it is sufficient to provide python code that the agent can call to examine the quality of its current solution, such as a model checkpoint or an RL algorithm. In contrast, MLE-Bench requires a CSV file to be submitted for grading each question and SWE-Bench/Agent require evaluating a piece of code through a collection of unit tests. MLAgentBench, RE-Bench and ScienceAgentBench provide Python scripts to compute the evaluation scores.

Finally, MLGym enables easy evaluation of both models and agents. To facilitate model evaluation, MLGym provides a default agentic harness that can be used out-of-the-box to evaluate any base model. See Appendix E for a more extensive discussion of related work on LLM agents for coding, data science, and scientific research.

3 Capability Levels for AI Research Agents

We introduce a new framework for categorizing the capabilities of LLM agents at doing AI research. This consists of six levels, each representing a distinct degree of autonomy and scientific contribution .

Level 0: Reproduction At this level, LLM agents can reproduce existing research papers either with or without access to the original code. This level demonstrates a basic understanding of the research domain and the ability to replicate established results.

Level 1: Baseline Improvement At Level 1, LLM agents can improve performance on a benchmark given a baseline code that is not state-of-the-art (SOTA). This level indicates the ability to analyze and optimize existing solutions, even if they are not the most advanced.
Level 2: SOTA Achievement At Level 2, LLM agents can achieve SOTA performance on a benchmark given only a task description and access to the published literature before the invention of the SOTA approach, but no access to the SOTA paper or code. This level demonstrates the ability to come up with a solution to an open research problem which is as good as the one found by humans.

Level 3: Novel Scientific Contribution At Level 3, LLM agents can make a novel scientific contribution, such as coming up with a new method that establishes a new SOTA on multiple benchmarks, and is worthy of publication at a top ML conference such as NeurIPS.

Level 4: Groundbreaking Scientific Contribution At Level 4, LLM agents can identify key research questions, directions, solutions, and make a notable scientific contribution worthy of being published as an oral or best paper award at a prestigious ML conference such as NeurIPS.

Level 5: Long-Term Research Agenda At Level 5, LLM agents can pursue a long-term research agenda, coming up with the research questions, directions, and solutions, continuously producing scientific discoveries over the span of weeks, months, or years. LLMs at this level should be capable of paradigm-shifting research breakthroughs worthy of prizes such as Nobel or Turing.

By defining these capability levels, we provide a framework for evaluating frontier AI Research Agents.

MLGym-Bench focuses on Level 1: Baseline Improvement of the categorisation defined above.

4 MLGym

An LLM agent can perform ML research/development by interacting with a shell environment through a sequence of commands. Given a task description, some starter code and access to its action and observation history, the LLM generates appropriate shell commands to accomplish research objectives like generating ideas, processing data, implementing new methods, training and evaluating models, analyzing the results, and reasoning about what experiments to run next. The agent is iteratively prompted to take actions based on the task description and execution feedback from previous commands, allowing it to develop and self-refine the solutions in-context.

The MLGym framework provides a unified framework for evaluating and developing agents and models for AI research tasks. We take inspiration from long existing field of RL and build a GYM (Brockman et al., 2016) environment that can execute shell commands in a local docker machine shell. MLGym **provides access to four core components: Agents, Environment, Datasets, and Tasks**. MLGym's modular design allows one to easily utilize and extend the library. For example, researchers can easily implement other agentic harnesses to improve performance, they can expand the environment by adding more tools for an agent, add more datasets within a given task (e.g., if the task is image classification they could add ImageNet in addition to Cifar-10), and they can even add more tasks to the MLGym benchmark. Below, we discuss each component in detail.

4.1 Agents

The Agent class provided by MLGym acts as a wrapper around a base LLM and provides functionality for integrating various base models, history processors, and cost management. Moreover, unlike other frameworks (Huang et al., 2024; Yang et al., 2024), MLGym separates the agent from the environment, allowing for easy integration of external agents. This also enables one to fairly compare different base models given the same agentic harness without the need of implementing their own agentic orchestration.

The agent is expected to take the history of all prior observations and actions as input and return the next action to take. The provided action is then passed to the environment, which executes the command and returns the next observation based on the command output. The agent can execute any BASH COMMAND in the environment. In addition, it has access to a set of tools (i.e., bash scripts such as editing a file) that it can use similarly to any other bash command. MLGym provides an agent adapted from SWE-Agent (Yang et al., 2024) as a default agentic harness. We describe the design and configuration of the tools in subsection 4.5. The full system prompt used can be found in subsection D.1.

4.2 Environment

MLGym environments are designed as Gymnasium (*gym*) environments (Towers et al., 2024). The environment component is responsible for initializing a *shell environment* in a local *docker machine*, with all the required tools, installing task-specific *python dependencies*, copying all the necessary data and code in a separate agent workspace and managing interactions between the LLM agent and the system. Moreover, to support open-ended research tasks and make the environment safe and flexible, MLGym environment also manages permissions for various files and directories. Specifically, when running in a docker container, due to various security concerns associated with using a root user, we create a non-root user named ”agent” and set the appropriate permissions for the working directory.

In this work, we make a conscious decision to decouple tools and ACI as defined in SWE-Agent (Yang et al., 2024)[2]. Note that this ensures that the agent and environment are not tightly coupled, allowing for easier implementation of other agentic architectures. Practically, this means that when the environment is initialized, it also initializes the tools in the working environment and a tool documentation is prepared which can be added to the LLM agent's prompt. More details about the tools are provided in subsection 4.5.

4.3 Datasets

MLGym provides a simple abstraction for defining datasets through configuration files. It supports both locally stored and Hugging Face datasets. We decouple the dataset definition from the task definition, so that a single dataset can be used in multiple tasks. Similarly, a single task can have more than one dataset so that the agent's code can be evaluated across all of them to demonstrate the generality of the implemented method.

Moreover, if the dataset files are stored locally, the environment automatically copies the relevant files to the agent workspace with read-only permissions. This ensures that the agent cannot change the dataset files, which is important for reproducibility and cheating prevention.

If the dataset is stored in Hugging Face, the agent is given the dataset URL through the starter code or in the prompt and asked to utilize it. Note that if the LLM agent fails to follow instructions or uses a different dataset, the evaluation code will not work or result in performance issues.

[2]As of the latest release, SWE-Agent also decouples tools/ACI from the agent.

4.4 Tasks

We provide an easy abstraction to define any ML research task using configuration files. Each task can incorporate one or more datasets, custom evaluation scripts (with read-only access), task-specific conda environment, optional starter code, training timeouts, and memory management settings. This provides a flexible framework for defining diverse open-ended ML research tasks covering a wide range of difficulty. For example, one can define an easier version of a task by providing a baseline code and a harder version by providing no starter code or one with bugs, thus creating a natural curriculum.

Evaluation is a critical component for any ML task. Every task requires a different evaluation protocol; thus, Kaggle-style evaluation as done in MLE-Bench (Chan et al., 2024) where the agent is expected to submit a CSV file with model predictions for a set of inputs is not feasible for every problem. For example, in reinforcement learning settings, the evaluation artifact is a set of models trained on a set of pre-defined random seeds, which is then used to get a mean reward across a set of environment seeds. Similarly for Game Theoretic tasks, it can be a Python file with a strategy function which will be evaluated against a fixed set of strategy functions. Since we aim to evaluate the agent on open-ended and diverse tasks, it is not possible to convert all submissions to a CSV format with model predictions for a set of inputs. To ensure extensibility to such open-ended tasks, the task definition is expected to provide an evaluation script and submission artifact instructions. The LLM agent can then be prompted to follow the submission instructions and write the appropriate code. Moreover, the evaluation script is read-only for the LM agent, so while it can inspect the evaluation format, it cannot modify the script to change the evaluation logic.

Existing works such as Huang et al. (2024); METR (2024); Chen et al. (2024) also use a script based evaluation approach, whereas MLE-Bench (Chan et al., 2024) uses a Kaggle style evaluation.

Our design decisions for the Agent, Environment, Dataset, and Tasks aim to reduce overhead for developers and researchers, and enhance reproducibility in this newly emerging area.

4.5 Tools and ACI

Augmenting LLM agents with the ability of using external tools is a critical component for making progress on knowledge-intensive tasks. In this work, we extend the ACI (agent-computer interface) first introduced in SWE-Agent (Yang et al., 2024) with some additional features required for an ML research agent. Specifically, we extend the commands for search, navigation, file viewer, file editor and context management with our permission management system and introduce new commands for literature search and a memory module. For example, if the agent tries to open a file without read permission, the file viewer tool will generate textual feedback for the agent. Similarly, if agent tries to edit the evaluation script (which is marked as read-only), the edit tools will output a feedback string instead of failing silently. Literature search and the ability to maintain a experimental log in it's memory are crucial for the agent to surpass SOTA solutions on open-ended research tasks.

Similar to SWE-Agent, tools are defined as bash or python scripts and are made available as bash commands in the environment.

All tool documentation is provided to the agent in the system prompt. See Table 2 for a short description of the available tools and Appendix B for more details. For all the experiments presented in this paper, the agent only uses the SWE-Agent tools and validation command.

5 MLGym-Bench

The primary motivation behind our benchmark is to challenge models across different aspects of machine learning, including data handling, model architecture, and strategic decision-making. By incorporating tasks from data science, game theory, computer vision,

Category	Tool	Arguments	Documentation
SWE-Agent Tools			
Search	**search_dir**	< search_term > [< dir >]	searches for the search term in all files in dir
	search_file	< search_term > [< file >]	searches for the search term in the given file
	find_file	< *file_name* > [< *dir* >]	finds all the files with the given name in dir
File Viewer	**open**	< path > [< line_number >]	opens the given file and goes to the line number
	goto	< line_number >	moves the window to show the line number
	scroll_down		moves the window down 1000 lines
	scroll_up		moves the window up 1000 lines
File editing	**create**	< filename >	creates a new file
	insert	< line_number < text_to_add >	inserts the given text at line number in the open file
	edit	< start_line >:< end_line < replacement_text >	replaces the given lines with the given text in the open file
Evaluation	**validate**		validates the current submission file and returns the metrics on the test set
	submit		submits the current code and terminates the session
Extended Tools			
Literature Search	**literature_search**	< query > [< num_results >]	query Semantic Scholar API for papers with attached PDFs
	parse_pdf_url	< url >	downloads and extracts the contents of a PDF given a URL
Memory Module	**memory_write**	< content_str >	save important results, configs or findings to memory
	memory_read	< query_str >	retrieve top-2 elements from memory most similar to a query

Table 2: List of tools available to agents. Required arguments are enclosed in <> and optional arguments are in [].

natural language processing, and reinforcement learning, the benchmark aims to provide a varied and comprehensive agent evaluation testbed.

The tasks included in the benchmark are carefully selected to represent real-world challenges, ensuring that models are tested on their ability to generalize and perform effectively across various scenarios. Each task is accompanied by standardized evaluation scripts and baseline implementations, providing a clear reference point for performance assessment and comparison.

The benchmark suite is structured into four main categories, each focusing on a specific domain of machine learning: Data Science, Game Theory, Computer Vision, Natural Language Processing, and Reinforcement Learning. See Appendix A for a detailed description of all the tasks.

6 Experimental Setup

For our experiments, we utilize a SWE-Agent (Yang et al., 2024) setup adapted specifically for the MLGym environment (see Appendix D). We use the SWE-Agent Tools as described in Table 2.

We evaluate a number of frontier LLMs for our experiments, `GPT-4o`, `o3-mini`, `o1-preview`, `Gemini 1.5 Pro`, `Gemini 2.0 Flash`, `Gemini 2.5 Pro`, `Claude-3.5-sonnet-20241022` (referred to as Claude-3.5-sonnet in the paper), `Claude-3.7-Sonnet`, `Llama3.1-405b-instruct`, `Llama4-Scout`, `Llama4-Maverick` and `DeepSeek-R1`. All the models are used with `temperature=0.0` and `top-p=0.95`, with the exception of o1-preview and o3-mini, which doesn't support changing the decoding parameters and has a default `temperature=1.0`.

A single agent run is limited to 50 steps (i.e. interactions with the environment) or $4 API cost limit, whichever occurs first, after which the agent is terminated and the last codebase state is autosubmitted. Moreover, to control the runtime of the agent and prevent it from simply increasing the number of parameters in the model, we set a task specific timeout for the training commands that can be found in Table 9.

7 Evaluation

In order to compare agents on MLGym, we aggregate the scores of each method—an agent architecture paired with a backbone model—across our tasks. Rather than naive averaging of scores or rankings which can lead to unfair comparisons, we employ performance profile curves (Dolan & Moré, 2002; Tu et al., 2022; Roberts et al., 2022b) and the area under these curves referred to as AUP scores (Roberts et al., 2022a; Dahl et al., 2023). These metrics capture relative performance gains across both methods and tasks and were originally

developed in the optimization and AutoML communities. Next, we define performance profiles, the AUP score, and the details of their usage within MLGym.

7.1 Performance Profiles and AUP Scores

For a given method m, its performance profile curve is defined as

$$\rho_m(\tau) = \frac{1}{|T|} \left| \{ t \in T : \log_{10} r_{t,m} \leq \tau \} \right| \qquad r_{t,m} = \frac{\ell_{t,m}}{\min\{\ell_{t,m} : m \in M\}} \tag{1}$$

where M is the set of all methods, P is the set of tasks, $\ell_{t,m}$ is the performance metric for a method m on task t, and $r_{t,m}$ is a quantity called the *performance ratio*.

This definition assumes that the performance metric for each task, $\ell_{p,\cdot}$ is better if lower—see subsection 7.2 for how to adapt it. Performance profiles are parameterized by a threshold, τ, on the distance between the method m and the best scoring methods on each of the tasks. At a given threshold τ, performance profiles compute *the proportion of tasks for which the method m is within τ of the best method for each task*.

In order to derive a final score for each method $m \in M$, we compute the AUP score as

$$\text{AUP}_m = \int_1^{\tau_{\max}} \rho_m(\tau) d\tau, \tag{2}$$

where $\tau_{\max}$ is the minimum τ for which $\rho_m(\tau) = 1$ for all $m \in M$.

7.2 Usage in MLGym

In the context of MLGym, a method is defined as a combination of an agent scaffolding and a backbone model. We adapt performance profiles and AUP scores to handle various edge cases introduced by our MLGym tasks. **Metric Direction Handling:** for metrics where higher values are better (e.g., accuracy, R2), we invert the performance ratio calculation and use the maximum score instead of the minimum: $r_{t,m} = \frac{\max\{\ell_{t,m} : m \in M\}}{\ell_{t,m}}$. **Infeasible Method:** in order to be counted as a feasible method, an agent should produce at least one valid solution and beat the baseline, methods must outperform the baseline. Methods that don't produce any valid solution or underperform are marked as *Infeasible*. The score of an infeasible method is set to $(1 + \varepsilon) \times r_{t,m_{\text{baseline}}}$, where $r_{t,m_{\text{baseline}}}$ is the score obtained by the baseline method on task t. We set the value of $\varepsilon = 0.05$.

We report the metrics across 4 independent runs for each model on each task. Finally, since the LLM agent can use the `validate` command to check the performance without ending the run, we maintain two separate sets of performance profiles and AUP scores for each model. ***Best Submission Profiles,*** $\rho_m^{\boldsymbol{bs}}(\tau)$**@4,** are computed using the best final submission across 4 runs. A submission is classified as a final submission in two cases: if the agent uses the `submit` command, or if the agent terminates without submitting and the last codebase state is used to evaluate performance. ***Best Attempt Profiles,*** $\rho_m^{\boldsymbol{ba}}(\tau)$**@4,** which are computed using the best attempt across 4 runs. Any valid call to the `validate` command is considered an attempt.

The resulting AUP scores provide complementary information. $\boldsymbol{AUP_m^{bs}}$**@4** indicates the model's ability to consistently submit its best attempt as the final solution. Note that to do this, the LLM agent has to be able to keep an internal state of the best attempt and recover from any mistakes made after the best attempt was made. $\boldsymbol{AUP_m^{ba}}$**@4** captures the model's exploration capability and is an indicator of the ceiling of the model's performance.

Apart from the AUP scores and performance profiles, we also report the raw performance scores for each model on each task. Similar to performance profiles, we categorize the raw scores in two sets: Best Submission@4 and Best Attempt@4.

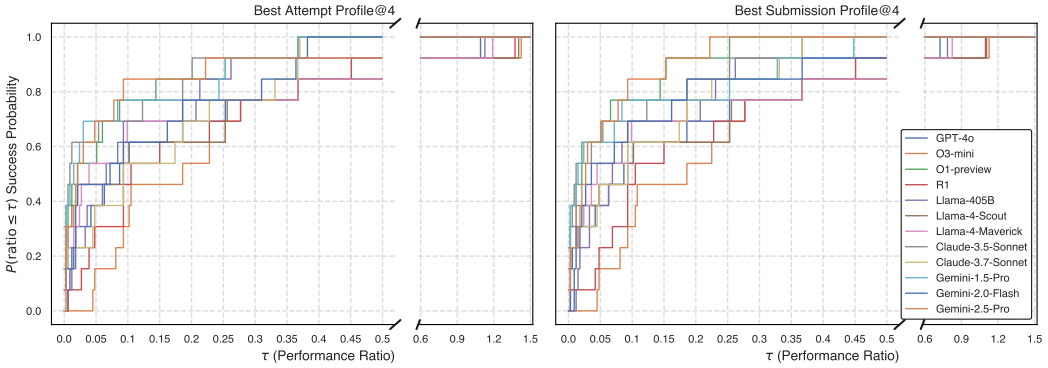

Figure 2: Performance profiles comparing Best Attempt@4 and Best Submission@4 across all models and tasks. The x-axis shows the performance ratio threshold τ and the y-axis shows the fraction of tasks where a model achieves performance within τ of the best model.

Model	Best Attempt AUP@4	Best Submission AUP@4
GPT-4o	1.288	1.317
o3-mini	1.214	1.236
o1-preview	1.423	1.444
DeepSeek-R1	1.249	1.267
Llama3.1-405b-instruct	1.330	1.353
Llama4-Scout	1.267	1.288
Llama4-Maverick	1.303	1.330
Claude-3.5-Sonnet	1.426	1.419
Claude-3.7-Sonnet	1.350	1.378
Gemini-1.5-Pro	1.420	1.405
Gemini-2.0-Flash	1.374	1.385
Gemini-2.5-Pro	1.419	1.445

Table 3: AUP@4 scores for the best attempt and best submission across all models. Best scores are highlighted in blue.

8 Results

8.1 AUP Scores and Performance Profiles

As detailed in section 7, we evaluate the performance of each model with the SWE-Agent scaffolding using Performance Profiles and Area Under the Performance Profile (AUP) scores. Figure 2 and Table 4 show the performance profiles and AUP scores, respectively, for the Best Attempt and Best Submission for all models.

In our experiments, we found that Claude-3.5-Sonnet and Gemini-2.5-Pro are the best-performing models on aggregate across our set of tasks for Best Attempt and Best Submission, respectively, with o1-preview and Gemini-1.5-Pro being close behind. See subsection C.1 for the performance scores of each model on each task.

8.2 Computational Cost

As discussed in Kapoor et al. (2024), it is important to also consider the pareto curve of performance vs cost for a more comprehensive evaluation of AI agents. Figure 3 shows the Best Attempt AUP@4 vs Average Cost for all models.

According to results discussed in subsection 8.1, OpenAI O1-Preview is the one of the best-performing models. However, it is also the most computationally expensive by a

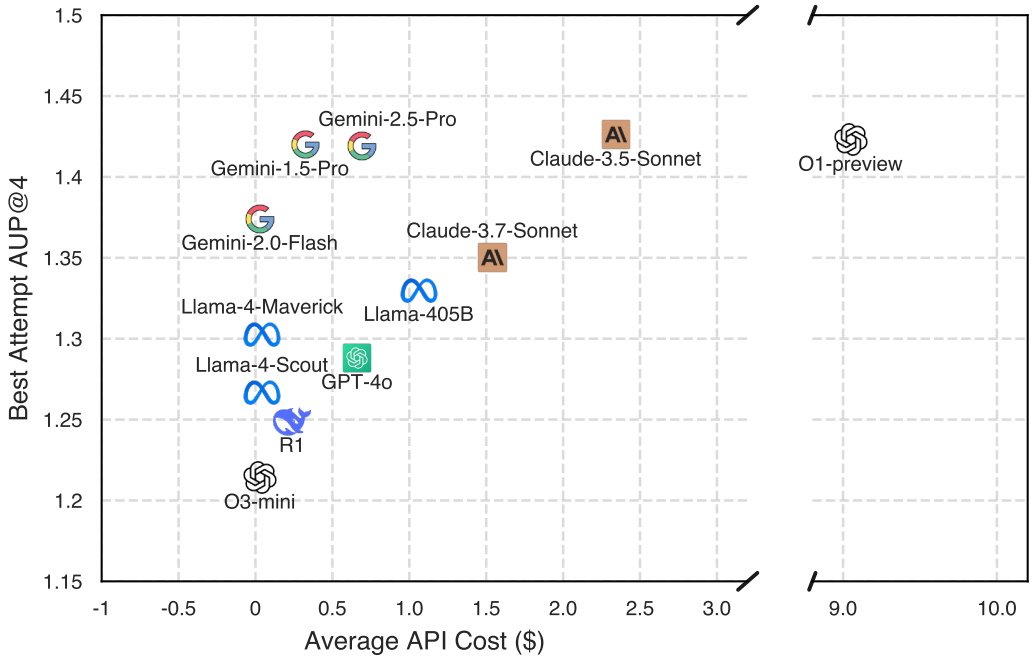

Figure 3: Best Attempt AUP@4 vs cost for all models. The x-axis shows the API cost in USD and the y-axis shows the AUP@4 score.

wide margin. In contrast, Gemini-2.5-Pro and Claude-3.5-Sonnet are much more cost-effective while surpassing O1-Preview performance. Gemini-1.5-Pro also shows impressive performance while being the most cost-effective.

Gemini-1.5-Pro is cheaper than both GPT-4o and Llama-3.1-405b-Instruct and provides massive performance gains relative to them. GPT-4o is one of the cheapest models to run but performs significantly worse than the top models, Claude-3.5-Sonnet, Gemini-2.5-Pro, and OpenAI O1-Preview. It is important to note that while DeepSeek-R1 and o3-mini are the cheapest models according to this plot, 5 shows that these two models have the highest rate of failures, thus likely terminating the runs early. Overall, Gemini series of models strikes the best balance between performance and cost on MLGym-Bench, being one of the cheapest models to run (approximately 9$\times$ cheaper than O1) while achieving at least 98% of Gemini-2.5-Pro's AUP (which is the top performing model).

For details on API pricing, tokens spent, and context length please consult Table 10. See subsection C.3 for an analysis of the agent behavior.

9 Conclusions

This paper presents MLGym and MLGym-Bench as initial steps toward building robust, flexible, and transparent LLM agents for AI research. As the field continues to evolve, improvements in long-context reasoning, better agent architectures, training and inference algorithms, as well as richer evaluation methodologies will be essential to fully harness LLMs' potential for scientific discovery, in general and for AI research in particular. By fostering collaboration among researchers in machine learning, scientific computing, and diverse application domains, we can move closer to a future where AI-driven agents meaningfully accelerate scientific research, all while maintaining verifiability, reproducibility, and integrity in scientific discovery. See Appendix G and Appendix H for more detailed discussions of the limitations and ethical considerations of our work, respectively.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1

AI Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1, 2024. 1

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement, 2024. URL https://arxiv.org/abs/2410.20285. 36

Robert Axelrod. Effective choice in the prisoner's dilemma. *Journal of conflict resolution*, 24 (1):3–25, 1980. 19

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. ResearchAgent: Iterative Research Idea Generation over Scientific Literature with Large Language Models, April 2024. URL https://arxiv.org/abs/2404.07738. 37

Ben Bogin, Kejuan Yang, Shashank Gupta, Kyle Richardson, Erin Bransom, Peter Clark, Ashish Sabharwal, and Tushar Khot. SUPER: Evaluating Agents on Setting Up and Executing Tasks from Research Repositories, September 2024. URL https://arxiv.org/abs/2409.07440v1. 2, 36

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540. 2, 4, 20

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, Victor Zhong, Lu Chen, Kai Yu, and Tao Yu. Spider2-v: How far are multimodal agents from automating data science and engineering workflows?, 2024. URL https://arxiv.org/abs/2407.10956. 36

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering, October 2024. URL https://arxiv.org/abs/2410.07095v1. 2, 3, 6, 37

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench: Toward Rigorous Assessment of Language Agents for Data-Driven Scientific Discovery, October 2024. URL https://arxiv.org/abs/2410.05080v1. 2, 3, 6, 37

Stephen A Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, 1971. 18

Russell Cooper, Douglas V DeJong, Robert Forsythe, and Thomas W Ross. Communication in the battle of the sexes game: some experimental results. *The RAND Journal of Economics*, pp. 568–587, 1989. 19

George Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, Juhan Bae, Justin Gilmer, Abel Peirson, Bilal Khan, Rohan Anil, Mike Rabbat, Shankar Krishnan, Daniel Snider, Ehsan Amid, and Peter Mattson. Benchmarking neural network training algorithms, 06 2023. 7

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL https://arxiv.org/abs/2306.06070. 2, 35

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 20

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, January 2002. ISSN 1436-4646. doi: 10.1007/s101070100263. URL https://arxiv.org/abs/cs/0102001. 2, 7

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 1

Katharina Eggensperger, Marius Lindauer, and Frank Hutter. Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893, 2019. 36

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. 36

M Emrich, A Agarwal, B Jairam, N Murthy, and OAK RIDGE NATIONAL LAB TN. Potential applications of artificial intelligence to the field of software engineering. Technical report, 1988. 1

Merrill M Flood. Some experimental games. *Management Science*, 5(1):5–26, 1958. 19

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024. URL https://arxiv.org/abs/2411.04468. 35

Drew Fudenberg and Jean Tirole. *Game theory*. MIT press, 1991. 18, 19

Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Abdelhakim Benechehab, Hamza Cherkaoui, Youssef Attia El-Hili, Kun Shao, Jianye Hao, Jun Yao, Balazs Kegl, Haitham Bou-Ammar, and Jun Wang. Large language models orchestrating structured reasoning achieve kaggle grandmaster level, 2024. URL https://arxiv.org/abs/2411.03562. 36

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752. 39

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zongze Xu, and Chenglin Wu. Data Interpreter: An LLM Agent For Data Science, March 2024. URL https://arxiv.org/abs/2402.18679. 36

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation, April 2024. URL https://arxiv.org/abs/2310.03302. 2, 3, 5, 6, 37

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. DISCOVERYWORLD: A Virtual Environment for Developing and Evaluating Automated Scientific Discovery Agents, June 2024. URL https://arxiv.org/abs/2406.06769. 37

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023. 2

Leland Johnson and Daniel Schaffer. *Oak Ridge National Laboratory: the first fifty years*. Univ. of Tennessee Press, 1994. 1

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024. URL https://github.com/KellerJordan/modded-nanogpt. 20

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*, 2023. 35

Kaggle. House prices - advanced regression techniques. Online; accessed January 24, 2025, 2016. URL https://www.kaggle.com/c/house-prices-advanced-regression-techniques. 18

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter, 2024. URL https://arxiv.org/abs/2407.01502. 9, 35

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024a. 36

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents, 2024b. URL https://arxiv.org/abs/2407.01476. 36

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 19, 20

Robert Tjarko Lange. gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax. 20, 21

P Langley. *Scientific discovery: Computational explorations of the creative processes*. MIT press, 1987. 39

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows, 2024. URL https://arxiv.org/abs/2411.07763. 36

Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. Autokaggle: A multi-agent framework for autonomous data science competitions, 2024. URL https://arxiv.org/abs/2410.20424. 36

Jiaju Lin, Haoran Zhao, Aochi Zhang, Yiting Wu, Huqiuyue Ping, and Qin Chen. Agentsims: An open-source sandbox for large language model evaluation, 2023. URL https://arxiv.org/abs/2308.04026. 35

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014. 20

Marius Lindauer and Frank Hutter. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, 21(243):1–18, 2020. 36

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents. https://arxiv.org/abs/2308.03688v2, August 2023. 35

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery, August 2024. URL https://arxiv.org/abs/2408.06292. 1, 37

R Duncan Luce and Howard Raiffa. *Games and decisions: Introduction and critical survey*. Courier Corporation, 2012. 19

Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, Aixin Sun, Hany Awadalla, and Weizhu Chen. Sciagent: Tool-augmented language models for scientific reasoning, 2024. URL https://arxiv.org/abs/2402.11451. 2

METR. Evaluating frontier ai r&d capabilities of language model agents against human experts, 11 2024. URL https://metr.org/blog/2024-11-22-evaluating-r-d-capabilities-of-llms/. 2, 3, 6, 37

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for General AI Assistants, November 2023. URL https://arxiv.org/abs/2311.12983. 35

Thomas Miconi, Aditya Rawal, Jeff Clune, and Kenneth O. Stanley. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity, 2020. URL https://arxiv.org/abs/2002.10585. 20

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2022. URL https://arxiv.org/abs/2112.09332. 35

Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. Llmatic: neural architecture search via large language models and quality diversity optimization. In *proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1110–1118, 2024. 36

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. *arXiv preprint arXiv:2411.13543*, 2024. 35

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318, 2002. 20

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557. 20

Karl Popper. *The logic of scientific discovery*. Routledge, 2005. 39

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789. 35

Brian Roberson. The colonel blotto game. *Economic Theory*, 29(1):1–24, 2006. 19

Nicholas Roberts, Samuel Guo, Cong Xu, Ameet Talwalkar, David Lander, Lvfang Tao, Linhang Cai, Shuaicheng Niu, Jianyu Heng, Hongyang Qin, Minwen Deng, Johannes Hog, Alexander Pfefferle, Sushil Ammanaghatta Shivakumar, Arjun Krishnakumar, Yubo Wang, Rhea Sukthanker, Frank Hutter, Euxhen Hasanaj, Tien-Dung Le, Mikhail Khodak, Yuriy Nevmyvaka, Kashif Rasul, Frederic Sala, Anderson Schneider, Junhong Shen, and

Evan Sparks. Automl decathlon: Diverse tasks, modern methods, and efficiency at scale. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht (eds.), *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 151–170. PMLR, 28 Nov–09 Dec 2022a. URL https://proceedings.mlr.press/v220/roberts23a.html. 2, 7

Nicholas Roberts, Xintong Li, Tzu-Heng Huang, Dyah Adila, Spencer Schoenberg, Cheng-Yu Liu, Lauren Pick, Haotian Ma, Aws Albarghouthi, and Frederic Sala. AutoWS-bench-101: Benchmarking automated weak supervision with 100 labels. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022b. URL https://openreview.net/forum?id=nQZHEunntbJ. 7

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761. 35

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *CoRR*, abs/1802.03685, 2018. URL http://arxiv.org/abs/1802.03685. 18, 20

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can llms generate novel research ideas? a large-scale human study with 100+ nlp researchers, 2024. URL https://arxiv.org/abs/2409.04109. 37

Xiangru Tang, Yuliang Liu, Zefan Cai, Yanjun Shao, Junjie Lu, Yichi Zhang, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Liang Chen, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yin Fang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and Mark Gerstein. ML-Bench: Evaluating Large Language Models and Agents for Machine Learning Tasks on Repository-Level Code, June 2024. URL https://arxiv.org/abs/2311.09835. 36

Artificial Intelligence Task Team. Artifical intelligence and nuclear power. 1985. 1

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. 1

Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, et al. Automl in the age of large language models: Current challenges, future opportunities and risks. *arXiv preprint arXiv:2306.08107*, 2023. 36

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032. 5

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents, July 2024. URL https://arxiv.org/abs/2407.18901. 35

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=xUXTbq6gWsB. 7

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pp. 22084–22102. PMLR, 2022. 3

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. 2, 35

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6): 186345, December 2024a. ISSN 2095-2228, 2095-2236. doi: 10.1007/s11704-024-40231-1. 35

Lei Wang, Jingsen Zhang, Hao Yang, Zhiyuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, Jun Xu, Zhicheng Dou, Jun Wang, and Ji-Rong Wen. User behavior simulation with large language model based agents, 2024b. URL https://arxiv.org/abs/2306.02552. 35

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenDevin: An Open Platform for AI Software Developers as Generalist Agents, July 2024c. URL https://arxiv.org/abs/2407.16741. 35

Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, and Ryan Cotterell (eds.). *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.conll-babylm.0/. 3

Adina Williams, Nikita Nangia, and Samuel R Bowman. The multi-genre nli corpus. 2018. 20

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024. 2, 35

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based Software Engineering Agents, July 2024. URL https://arxiv.org/abs/2407.01489. 36

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 20

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering, May 2024. URL https://arxiv.org/abs/2405.15793. 2, 3, 5, 6, 7, 35

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X. 28

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can Web Agents Solve Realistic and Time-Consuming Tasks?, July 2024. URL https://arxiv.org/abs/2407.15711. 35

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments, 2019. URL https://arxiv.org/abs/1903.03176. 21

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning, 2025. URL https://arxiv.org/abs/2410.02052. 36

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models, 2024a. URL https://arxiv.org/abs/2307.02485. 35

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024b. 36

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. 35

A MLGym-Bench Tasks

Below we describe each of the tasks in MLGym-Bench.

A.1 Data Science

House Price Prediction (Kaggle, 2016) In the House Price Prediction task, the goal is to predict housing prices using the Kaggle House Price dataset. This task evaluates models based on their ability to accurately predict prices from various features, using RMSE and R2 as performance metrics. The baseline for this task is a simple Ridge Regression model with minimal feature engineering.

A.2 3-SAT

3-SAT (Cook, 1971) In the 3-SAT task, the LLM agent is given a DPLL code and is prompted to optimize the variable selection heuristic. The associated DPLL code is stored in a read-only file, and the agent can inspect it to structure its heuristic function code, however, it cannot modify it. A simple random selection heuristic is used as a baseline and starter code for the LLM agent. The performance is measured by the total wall-clock time taken to solve a set of 100 generated 3-SAT instances. The instances are genereted using the algorithm described in Selsam et al. (2018).

A.3 Game Theory

We consider several tasks related to making strategic choices in iterated games, considering multiple well-known games. Specifically, we consider the task of producing code for a strategy for playing in a repeated two-player game. In each such task we provide an opponent strategy, in the form of an opponent bot for playing the game, and ask the agent to produce code for a strategy for best-responding to this opponent, i.e. provide code for a strategy that maximizes the score against that opponent. We very briefly review game theory terminology, with various textbooks covering this topic in more detail (Fudenberg & Tirole, 1991).

In a two-player **normal form game** G, players select actions simultaneously, with the outcome determined by the choices of both players. Let $A^1 = \{a_1^1, \ldots, a_k^1\}$ be the (pure) strategies available to player 1 and let $A^2 = \{a_1^2, \ldots, a_m^2\}$ be the strategies available to player 2. Denote the set of **strategy profiles**, consisting of a strategy choice for *both* players as $A = A_1 \times A_2$. The utility of the players depends on the actions selected by both for them, i.e. the payoffs are $u : A \to \mathbb{R}^n$, where $u(a) = (u_1(a), u_2(a))$ for $a \in A$, and where each player i tries to maximize their individual utility u_i. A mixed strategy is a probability distribution Δ over pure strategies. Given a mixed strategy profile $\sigma = (\sigma_1, \sigma_2)$ the expected utility of u_i of player i is $u_i(\sigma_1, \sigma_2) = \sum_{(a_1,a_2)\in A} \sigma_1(a_1)\sigma_2(a_2)u_i(a_1, a_2)$.

A repeated game consists of k rounds in which the players play the same underlying normal form game. The history at the $j+1$'th round consists of the actions (pure strategies) chosen by both players in each of the rounds 1 to j. We denote by H the set of all possible such histories, so a strategy in a repeated game is a function $a_i : H \to \Delta(A)$, i.e. a function that takes the history of actions chosen in the previous round and provides a distribution over the actions the agents would take in the next round. In our tasks, a strategy in the repeated game is expressed as a piece of code that takes in the history (actions of both players in the previous rounds), and outputs an action for the next round (where the code may make some random choices, hence yielding a distribution over the selected next round actions). Given an opponent strategy a_2, the goal of our agent is to produce a strategy that best responds to the opponent and produces a the maximal payoff, i.e $\arg\max_{a_1} u_1(a_1, a_2)$. Note that in this equation a_2 is a *given* opponent strategy expressed as a piece of code that takes the history over the previous rounds and selects an action for the next round (possibly making some random choices), and that the goal of an agent is to produce a_1 as a piece of code capturing the strategy of the first player. The agent optimization goal is selecting the code a_1 so as to maximize player 1's expected payoff u_1 against the fixed opponent a_2.

We consider the repeated version of prominent games, which we briefly discuss here: iterated Prisoner's Dilemma (Flood, 1958; Fudenberg & Tirole, 1991; Axelrod, 1980), Battle of the Sexes (Cooper et al., 1989; Luce & Raiffa, 2012) and Colonel Blotto (Roberson, 2006). As our goals was to highlight how our agent framework could be used to solve game theoretic tasks, rather than providing a rigorous evaluation and analysis of many game theoretic environments, we only included few games. However, additional games could easily be added in.

Prisonner's Dilemma (Axelrod, 1980). In this game, two players each have two options: cooperate or defect. When both cooperate, they receive a moderate reward. If one defects while the other cooperates, the defector gets a high reward while the cooperator gets a low payoff. If both defect, they both receive a low payoff. Due to the structure of payoffs, although mutual cooperation yields the best collective outcome, individual incentives often push towards defection. We included a repeated game, consisting of $k = 20$ rounds of the game. In the repeated version, players remember previous interactions and can adjust their strategies based on the history consisting of the past outcomes. Repeating the stage game multiple times allows for the development of trust and cooperation, as players recognize that consistent cooperation can lead to better long-term benefits than short-term defection (Axelrod, 1980). As our opponent strategy we provided a simple model which randomizes between cooperation, defection, or actions chosen based only on the last round of the interaction.

Battle of Sexes (Cooper et al., 1989). This is a simple game illustrating coordination challenges between two participants with different preferences. In the game, two participants have to agree on a venue (for instance where to go to spend an evening). There are two possible venues, and both players would rather make the same choice rather than making different choices. The strategic dilemma arises because as each player wants to coordinate their choice with the other, but they have a different ranking over the venues (one prefers the first venue and the other prefers the second venue). Similarly to the iterated Prisoner's Dilemma, we have used a repeated game with $k = 20$ rounds and used a simple opponent that makes random choices using the information from the last round.

Colonel Blotto Game (Roberson, 2006). This game is a model of strategic allocation of limited resources under competition. Two players ("Colonels") must simultaneously distribute their resources (such as troops) over several alternative locations ("battlefields"). The player who allocates more resources to a battlefield wins that battlefield. The overall winner is the player who wins the most battlefields. The key challenge arises from the fact that players must make their allocations without knowing how their opponent will distribute their resources. This yields an environment where players try and anticipate their opponent's moves to decide how to best allocate their own resources in order to maximize their chances of winning. A key insight from the game is the importance of diversification and unpredictability: it is harder to exploit an opponent who spreads resources across multiple battlefields and varies their strategy. Our target opponent used a very simple random allocation rule (re-normalizing to the overall budget of resources).

It is important to note that in all the game theoretic tasks, the agent is allowed to look at the opponent's strategy, and thus these tasks measure code understanding and the LLM's capabilities to exploit the opponent's strategy. In the future, we plan to add tasks where the opponent's strategy is not provided to the agent, and agent is pitted against multiple opponents in a round robin fashion, similar to the setup used in Axelrod's original Prisoner's Dilemma tournament.

A.4 Computer Vision

Image Classification (CIFAR-10) (Krizhevsky et al., 2009) The Image Classification CIFAR-10 task involves classifying images into one of ten classes using the CIFAR-10 dataset. This task tests the ability of models to learn visual patterns and features, with a baseline accuracy of 49.71% encouraging improvements

[3] https://www.kaggle.com/datasets/yasserh/housing-prices-dataset

Problem Setting	Domain	Task	Dataset/Environment
Supervised Learning	Data Science	Regression	House Price Prediction[3]
Supervised Learning	Computer Vision	Image Classification	CIFAR-10 (Krizhevsky et al., 2009)
Supervised Learning	Computer Vision	Image Classification	Fashion MNIST (Xiao et al., 2017)
Supervised Learning	Computer Vision	Image Captioning	MS-COCO (Lin et al., 2014)
Supervised Learning	Natural Language Processing	Natural Language Inference	MNLI (Williams et al., 2018)
Self-Supervised Learning	Natural Language Processing	Language Modeling	FineWeb (Penedo et al., 2024)
Reinforcement Learning	Reinforcement Learning	MetaMaze Navigation	Gymnax (Lange, 2022)
Reinforcement Learning	Reinforcement Learning	MountainCar Continuous	Gymnax (Lange, 2022)
Reinforcement Learning	Reinforcement Learning	Breakout MinAtar	Gymnax (Lange, 2022)
Algorithmic Reasoning	Computer Science	3-SAT	Randomly Generated (Selsam et al., 2018)
Algorithmic Reasoning	Game Theory	Prisonner's Dilemma	N/A
Algorithmic Reasoning	Game Theory	Battle of Sexes	N/A
Algorithmic Reasoning	Game Theory	Colonel Blotto	N/A

Table 4: List of tasks included in MLGym-Bench along with their respective problem setting, domain, and datasets.

Image Classification (Fashion MNIST) (Xiao et al., 2017) The Image Classification Fashion MNIST task involves classifying fashion items into predefined categories using the Fashion MNIST dataset. The agent is provided with a simple two layer CNN as a baseline and it has to optimize for the accuracy on the test set. The agent can optimize the model architecture and the hyper-parameters for the training.

Image captioning (MS-COCO) (Lin et al., 2014) For the image captioning task, the agent has to write the modeling code and come up with a good architecture and training setup for the image-text pairs in the MS-COCO dataset. We provide a baseline code for training to the agent which uses an image encoder and text decoder. We use the MS-COCO training and validation sets after removing all images containing humans. The agent has to optimize for the BLEU scores (Papineni et al., 2002) computed over the model-generated captions and ground truth captions for a given image.

A.5 Natural Language Processing

For language, we test the agent's ability to understand and modify training setup for both Natural Language Understanding (NLU) and Natural Language Generation (NLG) as detailed below.

Natural Language Inference (Williams et al., 2018) In this task, the agent starts from a pre-trained BERT model (Devlin, 2018) and we provide the baseline code to fine-tune on the training set of the MNLI benchmark to the agent. The agent is expected to come up with good hyper-parameters and fine-tuning strategy to optimize the test set accuracy on MNLI.

Language Modeling (Jordan et al., 2024) In the Language Modeling task, the agent is expected to train a language model for next token prediction using a smaller version of the FineWeb (Penedo et al., 2024) dataset. The LLM Agent is provided with the dataset and the NanoGPT (Jordan et al., 2024) codebase as a baseline and starting point. We use version `#8` from `modded-nanogpt`[4] as the starting point. The training and validation sets contain 1.773B and 100M tokens, respectively. The perfomance metric is the perplexity of the trained model on the validation set.

A.6 Reinforcement Learning

MetaMaze Navigation (Miconi et al., 2020) The MetaMaze Navigation task simulates a grid-world environment where agents must navigate using local observations and reach the goal location.

Mountain Car Continuous (Brockman et al., 2016) We use the continuous version of the Mountain Car environment introduced in Brockman et al. (2016), where the task is to learn a policy that drives a car up a steep hill in a continuous control environment.

[4] https://github.com/KellerJordan/modded-nanogpt

Breakout MinAtar (Young & Tian, 2019) The Breakout MinAtar task involves playing the arcade game Breakout in a simulated environment. This environment was introduced in Young & Tian (2019) and is a popular benchmark for evaluating reinforcement learning agents.

For all the RL tasks, we use the environments from the Gymnax library (Lange, 2022) and the PPO algorithm from Gymnax-blines[5] as a baseline and starting code for the LLM agent.

B MLGym Tools

Validation and Submit We provide two commands to the agent to validate the submission and submit the results. Both the validate and submit commands are used to run the evaluation script and give the agent feedback on its current score on the test set. However, while the submit command is a terminal action, i.e., the agent's trajectory is terminated, and the evaluation script is executed to log the final scores, the validate command can be used as many times as needed during the run to get the current performance on the test set.

Addition of a validation command helps the agent to continuously improve its performance on the test set.

Literature Search and PDF Parser We provide the agent with two tools to find and extract knowledge from external sources. The Literature Search tool allows the agent to query the Semantic Scholar API to find research papers about a given query that have open-access PDFs available, and the PDF Parsing tool allows the agent to download PDFs and convert them into a text-based representation. The paper contents can be stored in the context window as well as the Memory Module for longer-term tasks. Combined, these two tools allow the agent to find and analyze research papers as part of its workflow. See Table 2 for more information about these tools and how they are called.

Memory Module - Research Logs We introduce the Memory Module for MLGym, an important tool to improve the performance of agents on long-horizon AI research tasks. The Memory Module enables the agent to persistently store critical findings and successful training configurations using a structured memory system, overcoming the challenge of limited context retention in long tasks. During our experiments, we observed that when the agent has access to the memory module, it can retrieve the best training configuration from memory and continue to iterate on it (see Figure 11 and Figure 12). Without the memory module, the agent's trajectory can become longer than the model's context length, thus not being able to retrieve the best configuration, effectively forgetting older experiments and only being able to locally iterate on recent configurations.

The module is equipped with two core functions: `memory_write` and `memory_read`. The `memory_write` function allows the agent to store key insights and effective configurations by saving text data along with its corresponding embeddings and tags in JSON format. In contrast, the `memory_read` method retrieves the top-k most relevant stored entries based on cosine similarity with a given query, allowing the agent to review past knowledge and iterate from previously successful configurations.

Empirical results demonstrate the positive impact of the Memory Module on long-horizon tasks. Agents equipped with the Memory Module were able to sustain progress over extended sequences of trials, reusing optimal configurations and findings to achieve superior results compared to agents limited by fixed context windows. To further enhance its capabilities, we added the state of the memory to the system prompt (memory tags and number of records) so that the agent is aware of the type of data stored. Tags from a memory record are extracted by identifying the 3-gram most closely matching to the memory record.

This module significantly reduces the limitations of constrained context length, allowing agents to operate effectively in long experimental settings. However, it is an early version and there are many ways to improve the module. For example, one possible direction would be to introduce a more structured memory format, such as hierarchical or relational

[5] https://github.com/RobertTLange/gymnax-blines

Task	Metric	Baseline	DeepSeek-R1	Claude-3.5-Sonnet	Claude-3.7-Sonnet	Gemini-1.5-Pro	Gemini-2.0-Flash	Gemini-2.5-Pro
House Price Prediction	R^2 Score	0.88	∞	0.921	∞	0.914	0.91	0.9
3-SAT Heuristic	Wall-Clock Time (s)	16.158	12.612	15.728	14.677	14.36	12.383	13.207
CIFAR-10	Accuracy	0.497	∞	0.895	0.886	0.84	0.76	0.856
Fashion MNIST	Accuracy	0.783	∞	0.945	0.942	0.916	0.902	0.907
MS-COCO	BLEU Score	0.279	0.28	0.298	0.317	0.131	0.327	0.284
Language Modeling	Validation Loss	4.673	∞	4.476	∞	4.166	4.687	∞
MNLI	Validation Accuracy	0.525	0.834	0.830	∞	0.838	0.838	0.842
Battle of Sexes	Average Reward	1.023	1.024	1.442	1.443	1.443	1.445	1.442
Prisoners Dilemma	Average Reward	2.372	2.421	2.567	2.38	2.63	2.534	2.633
Blotto	Average Reward	-0.248	0.025	0.576	0.246	0.249	0.25	0.176
Breakout	Average Score	48.817	57.977	35.017	22.515	71.389	36.376	59.77
Meta Maze	Average Return	15.734	17.3	48.562	22.825	27.859	20.278	48.428
Mountain Car Continuous	Average Reward	33.794	$-\infty$	∞	∞	74.737	36.794	69.99

Table 5: Best Attempt@4 scores for Gemini, Claude, and DeepSeek models. Best scores across all models are highlighted in blue. *Note: ∞ indicates that the model was not able to produce even a single valid solution for submission or validation.*

Task	Metric	Baseline	GPT-4o	o3-mini	o1-preview	Llama3.1-405b	Llama4-Scout	Llama4-Maverick
House Price Prediction	R^2 Score	0.88	0.895	∞	0.931	0.908	0.892	0.903
3-SAT Heuristic	Wall-Clock Time (s)	16.158	13.676	14.244	13.652	13.793	11.894	12.626
CIFAR-10	Accuracy	0.497	0.733	0.1	0.857	0.548	0.499	0.716
Fashion MNIST	Accuracy	0.783	0.927	∞	0.92	0.876	0.922	0.896
MS-COCO	BLEU Score	0.279	0.176	∞	0.135	0.294	∞	0.473
Language Modeling	Validation Loss	4.673	4.361	∞	3.966	∞	∞	∞
MNLI	Validation Accuracy	0.525	0.819	∞	0.836	0.777	0.836	0.833
Battle of Sexes	Average Reward	1.023	1.149	1.15	1.444	1.261	1.442	1.442
Prisoners Dilemma	Average Reward	2.372	2.6	2.386	2.629	2.632	2.634	2.421
Blotto	Average Reward	-0.248	0.047	-0.244	0.248	0.043	0.023	0.037
Breakout	Average Score	48.817	0.0	∞	63.518	58.87	56.5	42.338
Meta Maze	Average Return	15.734	7.823	∞	34.986	26.744	0.703	7.877
Mountain Car Continuous	Average Reward	33.794	9.137	∞	61.277	71.726	∞	$-\infty$

Table 6: Best Attempt@4 scores for Llama and OpenAI models. Best scores across all models are highlighted in blue. *Note: ∞ indicates that the model was not able to produce even a single valid solution for submission or validation.*

models, allowing for precise storage and retrieval of information and enabling more complex reasoning over stored knowledge. Another is to incorporate memory operations directly into the model's training or fine-tuning process to allow the agent to natively utilize stored knowledge for improved performance. Or using a sub-agent that will automatically manage the memory by selecting important insights, removing unnecessary entries, and updating the memory. Each of these directions would require extensive experimentation and rigorous testing to ensure robustness and scalability.

C Additional Results and Analysis

C.1 Raw Performance Scores

To compare the performance of each model on each task, we also report aggregate metrics over 4 runs with different seeds, namely the Best Attempt@4 and Best Submission@4 in Table 5, Table 6 and Table 7, Table 8 respectively.

While Gemini-2.5-Pro is not dominant in most tasks, it beats out all the other models on aggregated scores due to the consistently being one of the top-performing models. Gemini-1.5-Pro, Claude-3.5-Sonnet, o1-preview, and Llama4 models occasionally taking the lead, it is consistently in the top-performing models for most tasks and thus takes the top spot in the AUP scores and performance profiles. This shows that the performance profile is a good metric to compare the performance of different models on a set of tasks with a diverse set of metrics.

Surprisingly, o3-mini struggles on most tasks while being one of the advanced models, but it is not the only one. Except for o1-preview, Gemini-1.5-Pro, and Gemini-2.0-Flash, all models fail to produce any valid submission on at least one task.

Task	Metric	Baseline	DeepSeek-R1	Claude-3.5-Sonnet	Claude-3.7-Sonnet	Gemini-1.5-Pro	Gemini-2.0-Flash	Gemini-2.5-Pro
House Price Prediction	R^2 Score	0.88	13.075	0.912	∞	0.908	0.91	0.9
3-SAT Heuristic	Wall-Clock Time (s)	16.158	13.075	15.728	14.677	14.36	12.604	13.326
CIFAR-10	Accuracy	0.497	∞	0.894	0.886	0.758	0.76	0.856
Fashion MNIST	Accuracy	0.783	∞	0.945	0.939	0.916	0.902	0.907
MS-COCO	BLEU Score	0.279	0.28	0.125	0.317	0.131	0.327	0.284
Language Modeling	Validation Loss	4.673	∞	4.476	∞	4.166	4.687	∞
MNLI	Validation Accuracy	0.525	0.834	0.830	∞	0.838	0.838	0.839
Battle of Sexes	Average Reward	1.023	1.024	1.439	1.442	1.443	1.44	1.442
Prisoners Dilemma	Average Reward	2.372	2.371	2.563	2.38	2.63	2.435	2.632
Blotto	Average Reward	-0.248	0.02	0.228	0.241	0.088	0.245	0.174
Breakout	Average Score	48.817	57.977	17.735	22.515	71.389	36.376	59.77
Meta Maze	Average Return	15.734	17.3	48.562	22.825	22.889	1.553	45.741
Mountain Car Continuous	Average Reward	33.794	$-\infty$	∞	∞	74.588	-0.055	69.99

Table 7: Best Submission@4 scores for Gemini, Claude, and DeepSeek models. Best scores across all models are highlighted in blue. *Note: ∞ indicates that the model was not able to produce even a single valid solution for submission or validation.*

Task	Metric	Baseline	GPT-4o	o3-mini	o1-preview	Llama3.1-405b	Llama4-Scout	Llama4-Maverick
House Price Prediction	R^2 Score	0.88	0.895	∞	0.931	0.908	0.878	0.903
3-SAT Heuristic	Wall-Clock Time (s)	16.158	13.676	14.244	13.83	13.936	11.894	12.846
CIFAR-10	Accuracy	0.497	0.733	0.1	0.854	0.528	0.499	0.716
Fashion MNIST	Accuracy	0.783	0.927	∞	0.906	0.876	0.922	0.896
MS-COCO	BLEU Score	0.279	0.111	∞	0.135	0.294	∞	0.473
Language Modeling	Validation Loss	4.673	4.361	∞	3.966	∞	∞	∞
MNLI	Validation Accuracy	0.525	0.819	∞	0.836	0.777	0.836	0.833
Battle of Sexes	Average Reward	1.023	1.144	1.131	1.439	1.256	1.442	1.439
Prisoners Dilemma	Average Reward	2.372	2.582	2.386	2.571	2.562	2.634	2.383
Blotto	Average Reward	-0.248	0.047	-0.244	0.247	0.041	0.019	0.037
Breakout	Average Score	48.817	0.0	∞	63.518	58.87	56.5	42.338
Meta Maze	Average Return	15.734	7.823	∞	34.986	26.744	0.703	7.688
Mountain Car Continuous	Average Reward	33.794	9.137	∞	52.73	71.726	∞	$-\infty$

Table 8: Best Submission@4 scores for Llama and OpenAI models. Best scores across all models are highlighted in blue. *Note: ∞ indicates that the model was not able to produce even a single valid solution for submission or validation.*

Task	Training Timeout	GPUs/Agents	Average Agent Runtime	Baseline Runtime (mins)
CIFAR-10	30m	1	4h	15
Battle of Sexes	30m	0	30m	5
Prisoners Dilemma	30m	0	30m	5
Blotto	30m	0	30m	5
House Price Prediction	30m	1	1.5h	10
Fashion MNIST	30m	1	2h	10
MS-COCO	40m	1	5h	7
MNLI	40m	1	2h	22
Language Modeling	40m	2	4h	20
Breakout	30m	2	2h	15
Mountain Car Continuous	30m	2	2h	15
Meta Maze	30m	2	2h	15
3-SAT Heuristic	30m	0	30m	5

Table 9: Computational resources required for each task in MLGym-Bench.

C.2 Computational Cost

Table 9 lists the resources needed to run the agent on each task in MLGym-Bench. Each task has a set Training Timeout, which is used as the time limit for any python commands. Specifically, it is used to prevent the agent from continuously scaling the model parameters. Average agent runtime and Baseline runtime show the wall clock time for each agent run and the provided baseline code, respectively.

Table 10 lists the average input and output tokens and associated pricing for each model across all tasks in MLGym-Bench. We report the model pricing as listed by their respective providers. Llama Model pricings are taken from Together AI.

Gemini-1.5-Pro charges 2X for using the long-context capabilities, i.e for input and output exceeding 128K tokens. However, in our experiments, we do not observe Gemini using the long-context capabilities, so the final price is reported based on the normal pricing.

[7] https://www.together.ai/pricing

Model	Avg. Usage Input	Avg. Usage Output	Pricing Input	Pricing Output	Context Length	Avg. Cost
Llama3.1-405b-instruct*	300607	2451	3.50	3.50	128k	$1.06
Llama4-Scout*	221452	2327	3.50	3.50	128k	$0.04
Llama4-Maverick*	156464	1908	3.50	3.50	128k	$0.04
DeepSeek-R1	397939	8583	0.50	2.19	64k	$0.24
Claude-3.5-Sonnet	718254	12481	3.00	15.0	200k	$2.34
Claude-3.7-Sonnet	433895	15823	3.00	15.0	200k	$1.54
Gemini-1.5-Pro	254279	1428	1.25	5.00	2M	$0.32
Gemini-2.0-Flash-Thinking	264775	3346	0.10	0.40	1M	$0.03
Gemini-2.5-Pro	430141	15513	1.25	5.00	2M	$0.69
GPT-4o	254186	2404	2.50	10.0	128k	$0.66
o3-mini	14747	2624	1.00	4.00	128k	$0.03
OpenAI o1	365643	59397	15.0	60.0	128k	$9.05

Table 10: Model pricing, token usage, context length, and average cost details. Model Pricing is in USD per 1M tokens. Average Cost is in USD per run. *Llama3.1: FP8 endpoint by Together[7]

C.3 Agent Behavior Analysis

C.3.1 Failure Mode Analysis

In this section we analyze the failure modes of our agents on MLGym-Bench tasks, using three key perspectives: termination error distribution, failed or incomplete run rates, and task-specific failure patterns. We collect trajectories across 13 tasks and 12 models with 4 different seeds. This results in a total of 624 trajectories with 48 and 52 trajectories for each task and model, respectively.

Termination Errors Figure 4 shows the distribution of different causes for termination encountered by each model during task execution, as indicated by the first word of the error message. We categorize the errors into the following types: `context length exceeded`, `evaluation error`, `file permission error`, `cost limit exceeded`, `format error`, and `runtime error`.

First, we observe that almost all models encounter Evaluation Error and is generally the most frequent final error, accounting for 75% of all termination errors. Evaluation Error is generally triggered by missing submission artefacts or incorrect submission format at the last step or when the `submit` command is issued.

Llama4-Maverick, o1-preview, GPT-4o, and Gemini-1.5-Pro demonstrate superior error handling capabilities with the lowest overall error rates. However, it is interesting to note that the Gemini family of models are the most cost-effective model across all tasks, but still encounters the Cost Limit error most frequently among all models.

Failed and Incomplete Runs The failed and incomplete run analysis in Figure 5 reveals significant variations in model reliability. If an agent run fails with a termination error without producing any valid intermediate submission, we mark it as failed. Whereas, if the run fails with a termination error but produces a valid intermediate submission i.e., at least one score on the test set is obtained, we mark it as incomplete. Note that the model's submission does not have to beat the baseline to be considered a valid intermediate submission. We are not interested in the performance of the model's submission here, but rather the ability of the agent to produce a valid submission by following the given instructions.

o3-mini exhibits the highest failure rate, while Llama4-Maverick and Gemini-1.5-Pro achieve the best completion rates. While Claude-3.5-Sonnet and Gemini-2.5-Pro are among the top-performing models across all tasks (subsection 8.1), they comparatively have a high failure rate. Another interesting observation is that OpenAI O1-Preview has a high incompletion rate rate, but it often produces at least one valid solution for all tasks.

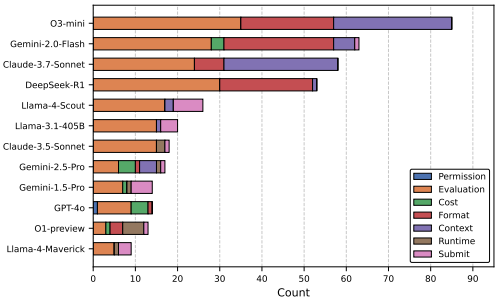

Figure 4: Termination Error Distribution by model. The size of the bars corresponds to the number of times each model triggered an exit status.

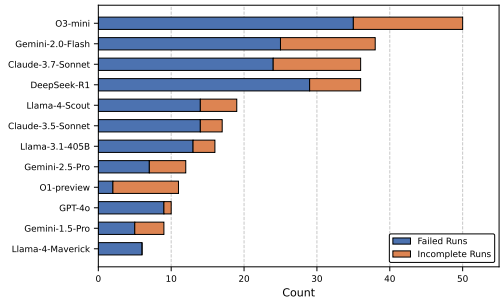

Figure 5: Number of Failed and Incomplete runs per model. The criteria for marking a run as incomplete or failed is described in subsubsection C.3.1

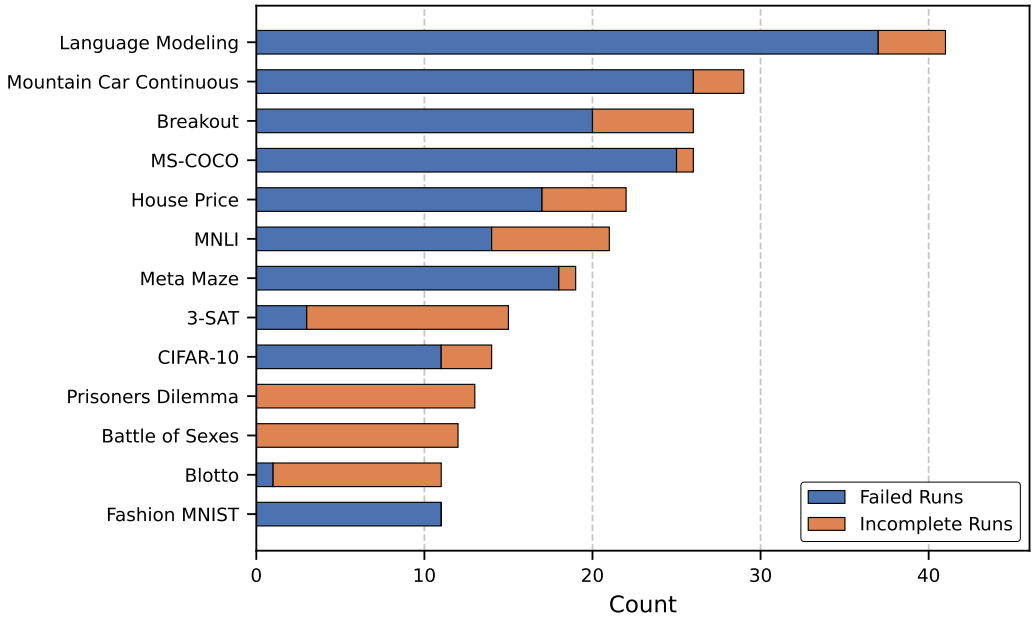

Figure 6: Number of Failed and Incomplete runs per task. The criteria for marking a run as incomplete or failed is described in subsubsection C.3.1

Figure 6 shows the failed and incomplete runs on each task to understand the difficulty distribution of tasks. Language Modeling and all Reinforcement Learning tasks (Meta Maze, Mountain Car Continuous and Breakout) prove the most challenging, with the highest failure rates. Whereas, Prisoner's Dilemma and Battle of Sexes show the lowest failure rates, with all models producing atleast one valid intermediate solution.

These failure patterns align with the raw performance scores in Table 5, Table 6 and Table 7, Table 8, where we observe that tasks requiring complex architectural decisions (Language Modeling) or complex algorithms (Breakout, Meta Maze and Mountain Car Continuous). Traditional supervised learning tasks are handled more reliably across models, while the more advanced models demonstrate better error handling and completion rates overall.

C.3.2 Action Analysis

In this section, we analyze the overall action distribution, as well as across models and trajectory steps. To analyze the action distribution effectively, we group the actions according

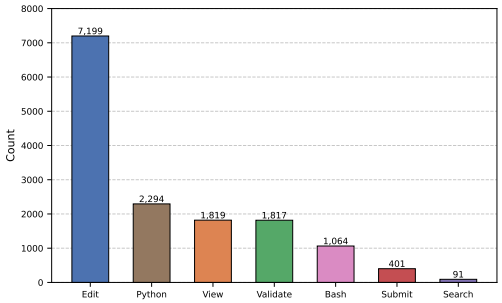

Figure 7: Action distribution across all runs. We group the actions into categories following the grouping defined in Table 2 and subsubsection C.3.2.

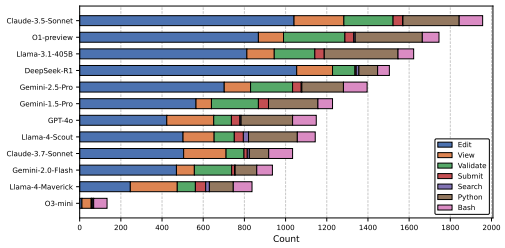

Figure 8: Action distribution for each model. We group the actions into categories following the grouping defined in Table 2 and subsubsection C.3.2.

to categories defined in Table 2: **Edit** , **View** , **Search** , **Validate** and **Submit** . We treat `validate` and `submit` as two separate categories.

Additionally, we have two open-ended categories: **Python** and **Bash** . All the actions that match the regex patterns `python.*`, `deepspeed.*`, `torchrun.*` are considered as **Python** actions. These actions usually correspond to the agent attempting to run a model evaluation or training script. All other actions are grouped under **Bash** category, i.e. are considered as open-ended bash commands.

Overall Action Distribution Figure 7 shows the action distribution across all runs. File commands such as **Edit** and **View** are one of the most frequently used commands with **Edit** accounting for 50% of the total actions. Whereas, **Search** commands are rarely used, accounting for only 1% of the total actions.

This distribution suggests that models spend a significant portion of their time in an iterative development cycle of editing and viewing files. Additionally, we observe a trend of regular experimental evaluation and periodic validation of solution by the frequent use of **Python** and **Validate** commands.

Per-Model Action Distribution Figure 8 shows the action distribution for each model. GPT-4o takes the least number of actions overall, indicating that the model either errors out or submits too early without reaching an optimal solution. This is consistent with the failure analysis shown in Figure 5.

Among the best-performing models, Claude-3.5-Sonnet and OpenAI O1-Preview perform the most number of actions within a run, while Gemini-1.5-Pro performs the least number of actions. Consistent with the cost analysis discussed in subsection 8.2, Gemini-1.5-Pro's lower trajectory length contributes to it being the most cost-effective model.

Per-Step Action Distribution Figure 9 illustrates the distribution of actions taken by agents across trajectory steps. Initially, **Bash** commands are predominant, indicating that agents start by checking and setting up their environment with basic commands such as `ls`, `pwd`, `cd` etc. As the steps progress, **Edit** actions become the most frequent, reflecting the agents' focus on modifying and refining code. This is complemented by a consistent use of **View** commands, suggesting a pattern of iterative development where agents frequently review their changes.

Python and **Validate** commands are used steadily throughout, which indicates an iterative cycle of experiments and evaluation. **Submit** actions are sparse, typically appearing towards the end of the process, aligning with the finalization of tasks. However, we can observe the **Submit** action being used as soon as Step 5, which indicates that some models submit their solution too early and likely fail to reach an optimal solution to beat other models.

Interestingly, **Search** commands are rarely used, suggesting that agents might benefit from improved search strategies to enhance efficiency while editing code.

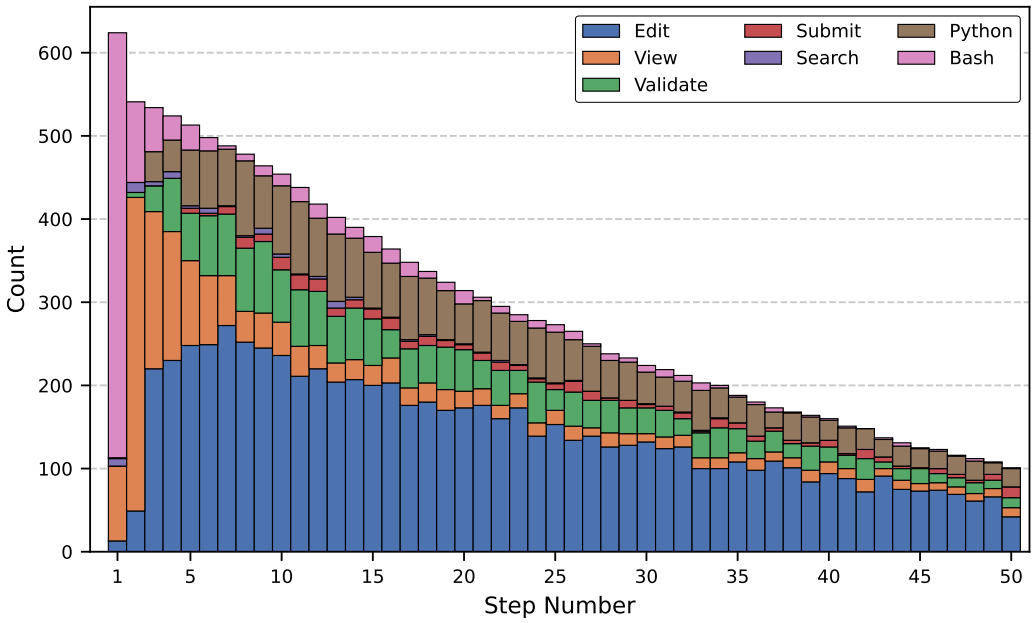

Figure 9: Action distribution for each step. We group the actions into categories following the grouping defined in Table 2 and subsubsection C.3.2.

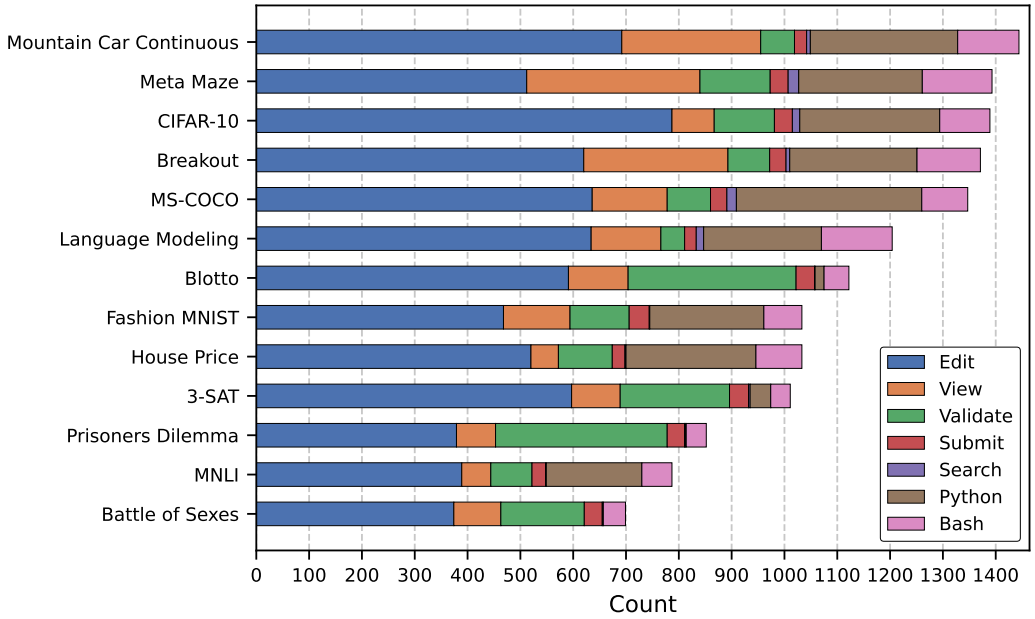

Figure 10: Action Distribution for each task. We group the actions into categories following the grouping defined in Table 2 and subsubsection C.3.2.

Overall, our analysis highlights a structured approach where agents begin with getting familiar with the environment and the task, conduct multiple iterations of experiments and validation, and conclude with and submission.

Figure Figure 10 shows the action distribution on each task. The bars represent the sum of all the actions taken by all models on a particular task. We notice that RL tasks have the

higest action count, while Game Theoretic tasks have the lowest action count. Algorithmic Tasks such as 3-SAT and Game Theory (Blotto, Prisonner's Dilemma and Battle of Sexes) also have the highest amount of validation actions, signifying a quick experimental cycle. Similarly, all RL tasks have the most complex codebases among all MLGym-Bench tasks and thus agent extensively use the View commands.

C.4 Memory Utilization

Figure 11 and Figure 12 show the agent using the memory module to store and retrieve specific experimental results and use them to submit the best possible model.

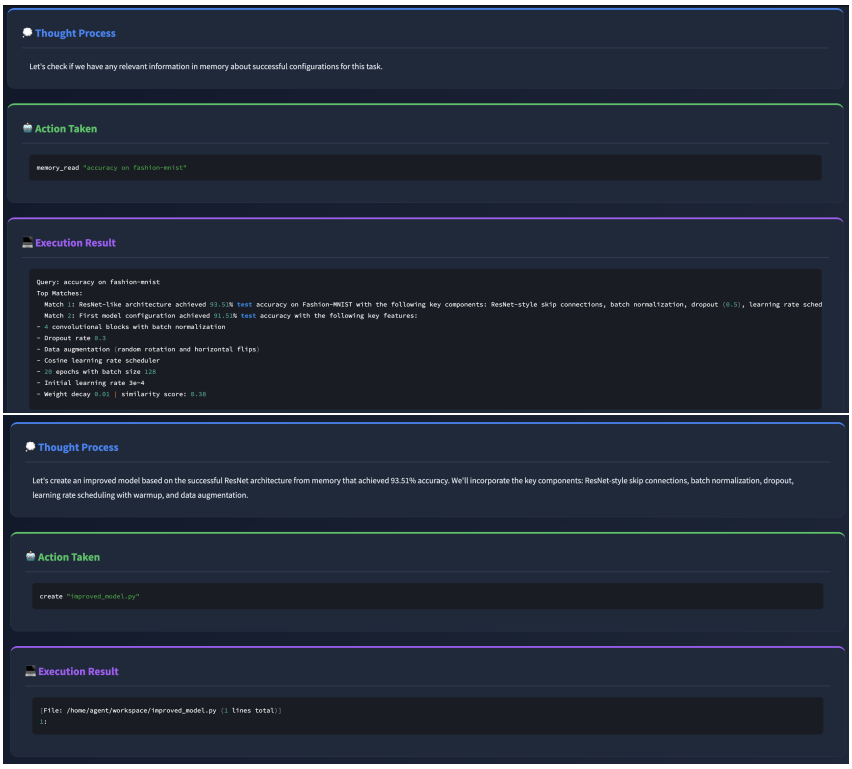

Figure 11: Example of retrieving the best training configuration from memory and restarting exploration from it.

D SWE-Agent Configuration

SWE-Agent follows a simple ReAct-style *thought and action* loop (Yao et al., 2023), where the agent is prompted with the ACI documentation, the task and dataset description, as well as lightweight generic instructions to act as a ML researcher. The agent is configured to use a single command per step, and is not allowed to use any interactive session commands (e.g., python REPL, vim). We discuss several key configuration settings of SWE-Agent below.

Window Size: Window Size parameter restricts the number of lines shown to the agent for viewer commands. The environment uses a window size of 1000 lines with an overlap of 2 lines, allowing the agent to effectively navigate and edit large files while maintaining context.

Context Management: A `History Processor` maintains the System Prompt, 1st User Message with Task Instructions, and a rolling window with the ***five most recent interactions (action and observation)***, helping the agent maintain context about the most recent interactions while keeping the input size manageable.

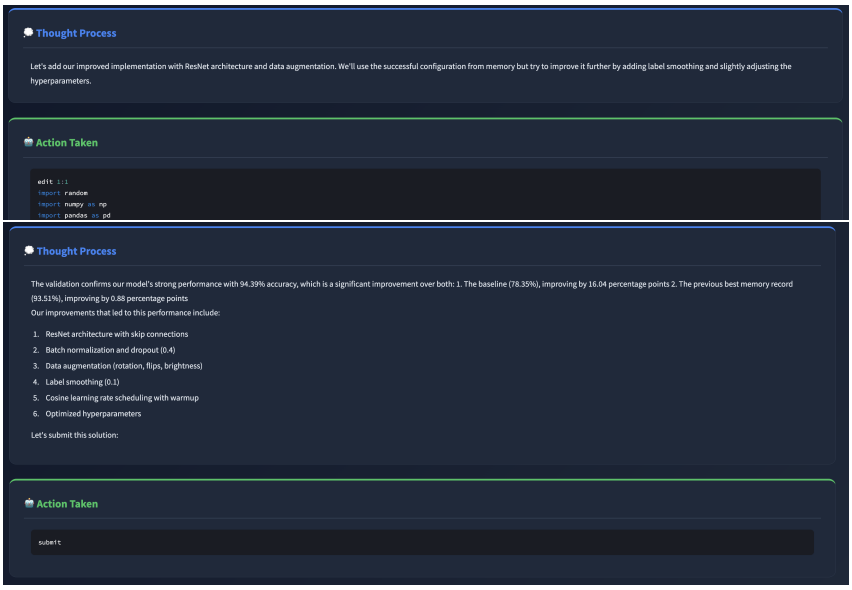

Figure 12: Example of retrieving the best training configuration from memory and restarting exploration from it.

Action Parser: The `Action Parser` parses the agent's output into a structured format. We prompt the agent to output a `Discussion` and a `single command` enclosed in ‘‘‘ and wait for a response from the shell before continuing with more discussion and commands.

Output Format

```
You need to format your output using two fields; discussion and command.
Your output should always include _one_ discussion and _one_ command
field EXACTLY as in the following example:

DISCUSSION
First I'll start by using ls to see what files are in the current
directory. Then maybe we can look at some relevant files to see what
they look like.

```
ls -a
```
```

Command Interface: The environment provides a set of specialized commands beyond standard bash operations, including file navigation commands (`goto`, `scroll_up`, `scroll_down`), file editing commands (`edit`, `insert`) with linting support, file and directory search commands (`search_file`, `search_dir`, `find_file`), and evaluation commands (`validate`, `submit`).

D.1 Prompt Templates

System Prompt

```
SETTING: You are an autonomous Machine Learning Researcher, and you're
working directly in the command line with a special interface.

The special interface consists of a file editor that shows you {WINDOW}
lines of a file at a time.
In addition to typical bash commands, you can also use the following
commands to help you navigate and edit files.

COMMANDS:
{command_docs}

Please note that THE EDIT and INSERT COMMANDS REQUIRES PROPER
INDENTATION.
If you'd like to add the line ' print(x)' you must fully write that out,
with all those spaces before the code! Indentation is important and code
that is not indented correctly will fail and require fixing before it
can be run.

RESPONSE FORMAT:
Your shell prompt is formatted as follows:
(Open file: <path>) <cwd> \$

You need to format your output using two fields; discussion and command.
Your output should always include _one_ discussion and _one_ command
field EXACTLY as in the following example:
DISCUSSION
First I'll start by using ls to see what files are in the current
directory. Then maybe we can look at some relevant files to see what
they look like.
```
ls -a
```

You should only include a *SINGLE* command in the command section and
then wait for a response from the shell before continuing with more
discussion and commands. Everything you include in the DISCUSSION
section will be saved for future reference. Please do not include any
DISCUSSION after your action.
If you'd like to issue two commands at once, PLEASE DO NOT DO THAT!
Please instead first submit just the first command, and then after
receiving a response you'll be able to issue the second command.
You're free to use any other bash commands you want (e.g. find, grep,
cat, ls, cd) in addition to the special commands listed above.
However, the environment does NOT support interactive session commands
(e.g. python, vim), so please do not invoke them.
Your goal is to achieve the best possible score, not just to submit your
first working solution. Consider strategies like validating your answer
using the `validate` command, manually spot-checking predictions,
building custom validation sets and grading functions, and comparing
different algorithms.
Once you have exhausted all possible solutions and cannot make progress,
you can submit your final solution by using `submit` command.
```

```
IMPORTANT TIPS:
1. Always start by trying to understand the baseline script if
available. This will give you an idea of one possible solution for the
task and the baseline scores that you have to beat.

2. If you run a command and it doesn't work, try running a different
command. A command that did not work once will not work the second time
unless you modify it!

3. If you open a file and need to get to an area around a specific line
that is not in the first {WINDOW} lines, don't just use the scroll_down
command multiple times. Instead, use the goto <line_number> command.
It's much quicker.

4. Always make sure to look at the currently open file and the current
working directory (which appears right after the currently open file).
The currently open file might be in a different directory than the
working directory! Note that some commands, such as 'create', open
files, so they might change the current open file.

5. When editing files, it is easy to accidentally specify a wrong line
number or to write code with incorrect indentation. Always check the
code after you issue an edit to make sure that it reflects what you
wanted to accomplish. If it didn't, issue another command to fix it.

6. You have a limited number of actions/steps you can take in the
environment. The current step and remaining number of steps will given
after every action. Use the remaining steps wisely. If you only have few
remaining steps, it is better to submit a working solution then to keep
trying.

7. Your each action should take less than {training_timeout} seconds to
complete. If your action doesn't finish within the time limit, it will
be interrupted.

8. Validating your solution often, will give you a good idea of your
progress so far and you will be able to adapt your strategy. Do not run
the evaluation file on your own, use the `validate` function instead. If
you run evaluation file yourself, your results won't be logged.
```

Command Docs We prepare command docs based on the YAML heredoc included in tool scripts. In this work, we use SWE-Agent tools as described in Table 2 and the documentation for the tools are given below.

Command Docs

```
open:
    docstring: opens the file at the given path in the editor. If
    line_number is provided, the window will be move to include that
    line
    signature: open "<path>" [<line_number>]
    arguments:
        - path (string) [required]: the path to the file to open
        - line_number (integer) [optional]: the line number to move
        the window to (if not provided, the window will start
```

```
        at the top of the file)

goto:
    docstring: moves the window to show <line_number>
    signature: goto <line_number>
    arguments:
        - line_number (integer) [required]: the line number
        to move the window to

scroll_down:
    docstring: moves the window down 1000 lines
    signature: scroll_down

scroll_up:
    docstring: moves the window down 1000 lines
    signature: scroll_up

create:
    docstring: creates and opens a new file with the given name
    signature: create <filename>
    arguments:
        - filename (string) [required]: the name of the
        file to create

search_dir:
    docstring: searches for search_term in all files in dir.
    If dir is not provided, searches in the current
    directory
    signature: search_dir <search_term> [<dir>]
    arguments:
        - search_term (string) [required]: the term to search for
        - dir (string) [optional]: the directory to search in (if
        not provided, searches in the current directory)

search_file:
    docstring: searches for search_term in file. If file is not
    provided, searches in the current open file
    signature: search_file <search_term> [<file>]
    arguments:
    - search_term (string) [required]: the term to search for
    - file (string) [optional]: the file to search in (if not
    provided, searches in the current open file)

find_file:
    docstring: finds all files with the given name in dir. If dir is
    not provided, searches in the current directory
    signature: find_file <file_name> [<dir>]
    arguments:
        - file_name (string) [required]: the name of the file
        to search for
        - dir (string) [optional]: the directory to search in
        (if not provided, searches in the current directory)

edit:
    docstring: replaces lines <start_line> through <end_line>
    (inclusive) with the given text in the open file. The replacement
    text is terminated by a line with only end_of_edit on it. All of
```

```
    the <replacement text> will be entered, so make sure your
    indentation is formatted properly. Python files will be
    checked for syntax errors after the edit. If the system
    detects a syntax error, the edit will not be executed.
    Simply try to edit the file again, but make sure to read
    the error message and modify the edit command you issue
    accordingly. Issuing the same command a second time will
    just lead to the same error message again.
    signature: edit <start_line>:<end_line> <replacement_text>
    end_of_edit
    arguments:
        - start_line (integer) [required]: the line number to start
        the edit at
        - end_line (integer) [required]: the line number to end the
        edit at (inclusive)
        - replacement_text (string) [required]: the text to
        replace the current selection with

insert:
    docstring: inserts the given text after the specified line
    number in the open file. The text to insert is terminated
    by a line with only end_of_insert on it. All of the
    <text_to_add> will be entered, so make sure your
    indentation is formatted properly. Python files will be
    checked for syntax errors after the insertion. If the
    system detects a syntax error, the insertion will not be
    executed. Simply try to insert again, but make sure to
    read the error message and modify the insert command you
    issue accordingly.
    signature: insert <line_number> <text_to_add>
    end_of_insert
    arguments:
        - line_number (integer) [required]: the line number
        after which to insert the text
        - text_to_add (string) [required]: the text to insert
        after the specified line

submit:
    docstring: submits your current code and terminates the
    session
    signature: submit

validate:
    docstring: validates your current submission file and
    returns the metrics on test set
    signature: validate
```

Task Template The task template is populated with task and dataset description and is used as the first `User Message` to the model.

Task Template

```
We're currently solving the following task. Here's the task description:

TASK DESCRIPTION:
```

```
{description}

INSTRUCTIONS:
Now, you're going to train a model to improve performance on this task.
Your terminal session has started and you're in the workspace root
directory. You can use any bash commands or the special interface to
help you. Edit all the file you need or create a new training script.
Remember, YOU CAN ONLY ENTER ONE COMMAND AT A TIME. You should always
wait for feedback after every command.
When you're satisfied with all of the changes you have made, you can run
your code. Your code should produce a valid submission artefact. Please
follow the instructions in SUBMISSION FORMAT section above and adhere to
the guidelines provided for generating submission artefacts. You can
also look at the `evaluate.py` provided to you to see if you are
following the correct format and naming scheme for your submission
artefacts.

Note however that you cannot use any interactive session commands (e.g.
python, vim) in this environment, but you can write scripts and run
them. E.g. you can write a python script and then run it with `python
<script_name>.py`.

NOTE ABOUT THE EDIT AND INSERT COMMANDs: Indentation really matters!
When editing a file, make sure to insert appropriate indentation before
each line!

(Current Step: {current_step}, Remaining Steps: {remaining_steps})
(Open file: {open_file})
(Current directory: {working_dir})
bash-\$
```

Next Step Templates These templates are used to prompt the agent for the next action generation. ***Next Step Template*** is used when the previous agent command results in an observation/output from the MLGym environment. Whereas, ***Next Step No Output Template*** is used when the agent command did not generate any output from the environment. The templates is populated with the following variables:

- `Observation`: Observation from the environment. This is only populated in the ***Next Step Template***.
- `Current Step`: Current step number for the agent
- `Remaining Steps`: Remaining steps for the agent before the code is autosubmitted.
- `Open File`: Currently open file. The agent can only edit/insert content in the currently opened file.
- `Working Dir`: Current directory for the agent. Equivalent to `pwd` in a Linux shell.

Next Step Template

```
{observation}
(Current Step: {current_step}, Remaining Steps: {remaining_steps})
(Open file: {open_file})
(Current directory: {working_dir})
bash-\$
```

Next Step No Output Template

```
Your command ran successfully and did not produce any output.
(Current Step: {current_step}, Remaining Steps: {remaining_steps})
(Open file: {open_file})
(Current directory: {working_dir})
bash-\$
```

E Extended Related Work

E.1 LLM Agents

Research on tool-augmented LLMs (Schick et al., 2023) has inspired a new research agenda of "agentic" LLMs (Kaddour et al., 2023; Wang et al., 2024a), where LLMs interact with an external environment. Existing work explores teaching LLMs to use tools or APIs (Schick et al., 2023; Qin et al., 2023), navigate the web (Nakano et al., 2022; Deng et al., 2023; Zhou et al., 2023), interface with operating systems (Wu et al., 2024), play games (Paglieri et al., 2024; Wang et al., 2023), or interact with other simulated (Wang et al., 2024b; Lin et al., 2023) or physical worlds (Zhang et al., 2024a). Evaluating agentic LLMs typically involves designing controlled environments, providing suitable tools, defining tasks and goals, and establishing quantitative metrics to measure the system's performance.

Building on these directions, Yoran et al. (2024) introduce *AssistantBench*, emphasizing the complexity of open-web navigation and showcasing how current systems struggle with realistic, time-consuming tasks such as monitoring real-estate markets or identifying nearby businesses. Meanwhile, Kapoor et al. (2024) highlight the importance of standardized evaluation protocols that consider both accuracy and cost, warning against overfitting and advocating for more reproducible benchmarks. Extending these concerns to multi-dimensional environments, Liu et al. (2023) propose *AgentBench*—a suite of eight interactive settings that test agents' capacity for reasoning, decision-making, and long-term instruction following. Similarly, Mialon et al. (2023) focus on holistic planning skills through *GAIA*, a benchmark designed to assess performance on real-world questions requiring robust tool-use and multimodal reasoning, revealing substantial gaps between human-level proficiency and current LLMs. Finally, Trivedi et al. (2024) emphasize the necessity of sophisticated tool integration with *AppWorld*, an interactive environment where agents must operate diverse applications via APIs and generate complex code in an iterative fashion. Collectively, these works underscore not only the breadth of agentic LLM capabilities but also the pressing need for systematic, multifaceted benchmarks that capture complex tasks with verifiable results and foster reproducible progress in the field. However, none of these works focuses on evaluating or developing LLM agents for open-ended AI research tasks.

E.2 Agents for Software Engineering and Data Science

In line with the principle of reproducibility and verifiability, software engineering tasks provide a testbed for LLM agents, where tasks can be tightly scoped and outcomes rigorously measured. Recent work has explored how agents can tackle code-level challenges in controlled settings that permit systematic evaluation. As discussed above, Yang et al. (2024) introduce *SWE-agent*, which operates within a constrained agent-computer interface to facilitate file creation, repository navigation, and code testing—thereby enhancing both traceability and reproducibility on benchmarks such as SWE-bench and HumanEvalFix. Similarly, Wang et al. (2024c) describe *OpenHands*, a platform that restricts agent interactions to sandboxed environments for safer command execution and verifiable web browsing, and in doing so provides a standardized foundation for benchmarking. Magentic-One (Fourney et al., 2024) is another agentic system competent in software engineering but also augmented with web navigation capabilities, as demonstrated by its strong performance on the GAIA, AssistantBench and WebArena (Zhou et al., 2023) agentic benchmarks. On the

other hand, Zhang et al. (2024b) achieve competitive perforemance on SWE-bench with AutoCodeRover, which, unlike the agentic approaches, solves Github issues by combining LLM-based programming with program representation as an abstract syntax tree.

Towards the goal of automating data science work, Li et al. (2024) introduce AutoKaggle, a multi-agent human-assisting system, and Grosnit et al. (2024) present AgentK v1.0, an end-to-end autonomous data science agent; both of these systems perform well on Kaggle competition data. Still within the realm of data science work, Lei et al. (2024) build Spider 2.0, a challenging benchmark and code agent framework for automating text-to-SQL workflows. Going one step further, Cao et al. (2024) introduce Spider 2-V, an autonomous multimodal agent coupled with a benchmark focusing on the automation of enterprise data science and engineering workflows.

More search-oriented approaches include *SWE-Search* (Antoniades et al., 2024), a multi-agent framework that marries Monte Carlo Tree Search (MCTS) with iterative refinement, enabling agents to continuously evaluate and improve their approaches to repository-level tasks. In a similar vein, Koh et al. (2024b) explore tree search for LLM agents and show that equipping LLM agents with best-first search boosts performane for the WebArena and VisualWebArena (Koh et al., 2024a) agentic benchmarks. Also on augmenting LLM agents with search, Yu et al. (2025) propose MCTS-based test-time search and self-learning techniques that yield better performance on VisualWebArena. Finally, Xia et al. (2024) demonstrate that even relatively simple approaches can excel when thoroughly monitored: an 'agentless' system follows a three-step process and outperforms more complex agent-based methods on SWE-bench Lite, underscoring the value of constrained, verifiable environments in driving reproducible gains for autonomous SWE agents.

E.3 Agents for Scientific Research

Controlled SWE contexts build the foundation for more complex automation while maintaining a reproducible and verifiable approach. However, just the software foundations alone are not sufficient to address the remaining gaps towards the goal of science acceleration. Going from the limited environments and well-defined tasks with metrics towards a less-defined area of open-ended questions, there are substantial efforts needed to boost the capabilities of research agents. For instance, coming up with automatable criteria to gauge scientific novelty or constructing theories inheriting the automated findings from heterogeneous disciplines are examples of areas that could use more refinement and experimentation.

Nevertheless, the first steps on this path can be started now - in the field of ML research and data science - since these areas represent for us a scientific playground with tasks that are both well-defined and have formal criteria of verifiability (benchmarks and tests), falsifiability (ablation studies and tests for data leakage, memorization, out of domain generalization, etc) and reproducibility.

Data Science. Many recent works approach both classic data science tasks and real-life repository-based tasks as a testbed for agents with a known test set and metrics. While based on similar grounds, the works differ in the resulting levels of autonomy of the agents. For instance, *ML-Bench* (Tang et al., 2024) focuses on explicit tasks within existing GitHub repositories — evaluating agents in code-centric setups without delving into open-ended objectives. By contrast, *Data Interpreter* (Hong et al., 2024) extends agent testing to broader data science problems, spanning coding tasks, mathematical reasoning, and a limited suite of open-ended applications (e.g., OCR, web search, and mini-game generation), thus reflecting a more flexible approach to autonomy. The agentic benchmark *SUPER* (Bogin et al., 2024) raises the bar by requiring the agent to formulate the task itself and iterate on NLP-related data and tasks within research repositories, thereby emphasizing self-directed problem-solving.

AI Research. The presence of models and simulations in machine learning itself inevitably leads to the fact that this area also becomes the object of automation. Having an agent formulating a task itself and approaching open-ended tasks naturally leads to automatic agentic enhancement of the machine learning methods themselves. AutoML (Eggensperger et al., 2019; Lindauer & Hutter, 2020; Tornede et al., 2023) and NAS (Elsken et al., 2019; Nasir

et al., 2024) approaches have been previously paving the foundations of ML automation within environments with built-in restrictions (an explicit set of methods, definition of the search space and strategy), while the agentic approach can propose open-ended solutions without said specifications.

For example, *MLAgentBench* (Huang et al., 2024) consists of an environment for agents to solve 13 complex tasks ranging from improving image classification to language modeling, with the current state-of-the-art LLMs achieving 0% success rate for the most difficult of these tasks. The proposed pipelines for agents in the environment include designing and running experiments, analyzing the results, and iterating towards improving the defined metrics. Similarly, *RE-Bench* (Research Engineering Benchmark) (METR, 2024) is a set of 7 diverse and challenging ML tasks with the methodological addition of real human experts involvement and progress comparison: timed sessions for ML experts vs LLM agents. Authors state that the best agents achieve a score 4x higher than human experts when both are given a total time budget of 2 hours per environment. However, humans currently display better returns to increased time budgets, narrowly exceeding the top AI agent scores given an 8-hour budget, and achieving 2x the score of the top agent when both are given 32 total hours. *MLE-bench* (Chan et al., 2024) focuses on Kaggle tasks as a source for agentic evaluations. Agents are evaluated across well-defined metrics, datasets, and real competition result distribution. The attempts are limited to 24 hours. However, in contrast with MLGYM, all these works contain a more narrow set of domains that do not assess algorithmic reasoning capabilities. Moreover, some of them do not provide a standardized agentic harness to allow for model evaluation, but they vary both the harnesses (also known as *scaffolds*) and the LLMs when comparing performances. While our work focuses on creating an evaluation framework with objective and standardized evaluation metrics, other recent works focus on developing an agentic harness for the more subjective task of generating papers based on end-to-end experimental cycles (Lu et al., 2024).

Scientific Discovery. Several recent works have approached scientific automation with LLM agents targeting the process of scientific discovery. *DiscoveryWorld* (Jansen et al., 2024) is a benchmark for scientific agents being evaluated in a game-like virtual discovery environment. 120 tasks require an agent to form hypotheses, design and run experiments, analyze results, and act on conclusions – for areas like proteomics, chemistry, archeology, physics, agriculture, rocket science, linguistics, or epidemiology. The custom simulation engine only supports a limited list of objects and 14 possible actions. A distinctive feature of the work is also that it focuses on general discovery skills rather than task-specific solution, and the assessment, space of objects and actions is common to all scientific domains.

ScienceAgentBench (Chen et al., 2024), however, approaches differently the similar task of creating a discovery-based agentic benchmark: the tasks are based on 44 cherry-picked peer-reviewed publications that include data-driven discovery tasks with well-defined metrics. The scientific areas covered include bioinformatics, computational chemistry, geographical information science, and neuroscience yielding 102 tasks of various types, such as data processing, modeling or visualization. Each task is defined by Python-based evaluation environment, end result metrics and intermediate evaluation criteria. Special metrics control data contamination and agent shortcut issues. Comparing different baselines, including pure LLMs with prompting, authors state that execution feedback is necessary for agents to generate useful solutions.

The idea of execution feedback and iterative improvement for research tasks has been proposed in *ResearchAgent* (Baek et al., 2024). Agentic concept-based approach with literature-based discovery shows great improvement for end-to-end iterative solution generation, also supported by knowledge-based vs random facts ablations. The agent is evaluated solely with subjective human preference annotation and automatic human preference evals. While covering structured aspects of end-to-end experimental pipeline (problem clarity, feasibility, significance, relevance, originality, method generalizability, innovativeness, experiment reproducibility, validity, etc), relying solely on human judgment without supporting it with objective metrics is insufficient, as Si et al. (2024) shows.

F Multi-Agent Research Evaluation

MLGym's modular architecture extends naturally to evaluate decomposed AI research systems where specialized agents collaborate on complex research tasks. Rather than relying on a single monolithic agent, this multi-agent approach divides the research process across specialized components: a literature review agent that synthesizes prior work, a hypothesis evaluation agent that assesses research directions, and an implementation agent that handles coding and experimentation.

F.1 Architectural Support for Specialized Agent Coordination

The framework's `BaseAgent` class and `ToolHandler` system provide the foundation for orchestrating multiple specialized agents. Each agent can be configured with distinct capabilities through the `AgentConfig` system, enabling role-specific tool access and behavioral patterns. For instance, a literature review agent could be equipped with the `literature_search` tool for querying research databases, while an implementation agent would have access to code editing and execution tools.

The `MLGymEnv` environment serves as the coordination layer, managing inter-agent communication and maintaining shared workspace state. The environment's container-based execution ensures that all agents operate within the same computational context while maintaining clear separation of responsibilities.

F.2 Evaluation Challenges in Decomposed Research

Evaluating multi-agent research systems presents unique challenges compared to single-agent approaches. The framework must assess not only the final research outcome but also the quality of intermediate products: literature synthesis, hypothesis formation, and implementation decisions. This requires developing evaluation metrics that capture:

Inter-agent Communication Quality: Measuring how effectively agents share insights, with successful handoffs requiring clear problem formulation from the research agent, actionable hypotheses from the evaluation agent, and interpretable results from the coding agent.

Specialization vs. Integration Trade-offs: Assessing whether task decomposition improves overall research quality compared to generalist approaches, particularly in scenarios where domain expertise significantly impacts research outcomes.

Coordination Overhead: Quantifying the computational and temporal costs of multi-agent coordination against potential improvements in research quality and reliability.

F.3 Framework Extensions for Multi-Agent Assessment

MLGym's task evaluation system can be extended to support multi-agent research assessment through several mechanisms:

Phase-specific Evaluation: The framework's `AbstractMLTask` interface can be enhanced to evaluate intermediate research products, such as literature summaries, hypothesis rankings, and implementation quality, providing detailed feedback on each specialist's contribution.

Collaborative Trajectory Analysis: The existing trajectory logging system can be expanded to capture inter-agent interactions, enabling analysis of communication patterns, decision dependencies, and collaborative problem-solving strategies.

Comparative Benchmarking: The standardized task interface allows for direct comparison between single-agent and multi-agent approaches across the same research problems, providing empirical evidence for the effectiveness of research decomposition.

F.4 Research Applications and Future Directions

This multi-agent evaluation capability enables investigation of fundamental questions in AI research automation: optimal task decomposition strategies, effective agent specialization patterns, and the scalability of collaborative AI research systems. By providing a standardized framework for evaluating decomposed research approaches, MLGym facilitates systematic study of how to best structure AI research teams and optimize the balance between specialist expertise and integrated problem-solving.

The framework's extensible design supports future enhancements such as dynamic agent composition, adaptive specialization based on task requirements, and hierarchical coordination structures for complex, multi-stage research projects.

G Discussion and Limitations

Our findings highlight both the opportunities and ongoing challenges in leveraging large language models (LLMs) as agents for scientific workflows. The proposed MLGym framework and accompanying MLGym-Bench tasks demonstrate that modern LLM agents can successfully tackle a diverse array of quantitative experiments, reflecting advanced skills and domain adaptability. At the same time, our results reveal notable capability gaps, which point to several avenues for improvement:

- **Scaling beyond ML tasks** To further evaluate the agent's AI Research capabilities, it is essential to scale up the evaluation framework to accommodate large-scale domain-specific datasets, more complex tasks, as well as domains outside AI. This will enable the community to assess the robustness and generalizability of different methods, as well as identify potential limitations and areas for improvement.
- **Interdisciplinary Ablations and Generalization** Within the stage of method evaluation, one approach is to test the solutions for generalization:
  - automatically evaluating the applicability of a new method on different domains . For example, new LLM architectures like Mamba (Gu & Dao, 2024) could be automatically applied to data on DNA, chemical molecules, music generation, etc.
  - automatically running interdisciplinary and multidisciplinary ablations, where we systematically remove or modify specific components of the proposed ML system to assess their impact on performance. This will enable the community to more quickly identify the most critical factors contributing to generalization across different domains.
- **Addressing Scientific Novelty** While the agentic benchmarks have demonstrated their effectiveness in evaluating complex tasks in different areas, it is essential to acknowledge that proposed interdisciplinary extrapolation of methods is just one aspect of the broader scientific understanding of "novelty" and "discovery" (Popper, 2005; Langley, 1987). It is not yet clear if the notion of scientific novelty can be successfully automated or even formally defined in a form suitable for agents. For many scientific disciplines, development may be uneven and depend on the availability of open data, the development of the methods, metrics and definitions used.
- **Data Openness Imperative** Finally, we emphasize the importance of data openness in driving scientific progress. By making our representative 'corpus of the world' widely accessible, including scientific artifacts, reproducible code, and domain-specific data for modeling, we can facilitate collaboration and accelerate discovery. This imperative is crucial for advancing our understanding of complex systems and developing more effective solutions to real-world problems. Removing once accessible resources that have entered LLM training from public access can have an irreparable impact on the acceleration of scientific progress, as it becomes impossible to identify sources of facts, and it is impossible to attribute the out-of-distribution result from a scientific work from a hallucination or a completely new result.

H Ethical Considerations

AI agents proficient in tackling open research challenges like those in our benchmark could catalyze a remarkable acceleration in scientific advancement. This prospect is exhilarating yet demands a meticulous comprehension of model progress to ensure responsible and controlled deployment of such breakthroughs. MLGym-Bench, for instance, can serve as a metric for model autonomy within OpenAI's Preparedness Framework, autonomous capabilities in Anthropic's Responsible Scaling Policy, and ML R&D in Google DeepMind's Frontier Safety Framework.

Should AI agents become adept at autonomously conducting AI research, the positive impacts could be multifaceted, encompassing accelerated scientific progress in healthcare, climate science, and other domains, expedited safety and alignment research for models, and economic growth spurred by the development of novel products. The ability of agents to deliver high-quality research could signify a transformative stride in the economy.

Nonetheless, agents capable of executing open-ended AI research tasks, such as enhancing their own training code, could augment the capabilities of cutting-edge models at a pace outstripping human researchers. If innovations outpace our ability to comprehend their ramifications, we risk developing models with catastrophic harm or misuse potential without parallel advancements in securing, aligning, and controlling such models. We believe a model proficient in solving a substantial portion of MLGym-Bench likely possesses the capacity to execute numerous open-ended AI tasks. We are open-sourcing MLGym and MLGym-Bench to foster understanding and research into the agentic capabilities of AI Research Agents and promote transparency regarding acceleration risks in frontier AI labs. In doing so, we acknowledge the limitations of MLGym-Bench and strongly encourage the development of additional evaluations of automated AI research capabilities, particularly those tailored to the workflow of researchers training frontier models.