AIRS-Bench: a Suite of Tasks for Frontier AI Research Science Agents

Alisia Lupidi[1,2,*], **Bhavul Gauri**[1,*], **Thomas Simon Foster**[1,2,*], **Bassel Al Omari**[1,*], **Despoina Magka**[1,*], **Alberto Pepe**[1], **Alexis Audran-Reiss**[1], **Muna Aghamelu**[1,†], **Nicolas Baldwin**[1], **Lucia Cipolina-Kun**[1], **Jean-Christophe Gagnon-Audet**[1], **Chee Hau Leow**[1], **Sandra Lefdal**[1], **Hossam Mossalam**[1], **Abhinav Moudgil**[1,†], **Saba Nazir**[1], **Emanuel Tewolde**[1,†], **Isabel Urrego**[1], **Jordi Armengol Estape**[1], **Amar Budhiraja**[1], **Gaurav Chaurasia**[1], **Abhishek Charnalia**[1], **Derek Dunfield**[1], **Karen Hambardzumyan**[1,3], **Daniel Izcovich**[1], **Martin Josifoski**[1], **Ishita Mediratta**[1], **Kelvin Niu**[1], **Parth Pathak**[1], **Michael Shvartsman**[1], **Edan Toledo**[1,3], **Anton Protopopov**[1], **Roberta Raileanu**[1,†], **Alexander Miller**[1], **Tatiana Shavrina**[1], **Jakob Foerster**[1,2], **Yoram Bachrach**[1]

[1]FAIR at Meta, [2]University of Oxford, [3]University College London

[*]Joint first author, [†]Work done at Meta

LLM agents hold significant promise for advancing scientific research. To accelerate this progress, we introduce AIRS-Bench (the *AI Research Science Benchmark*), a suite of 20 tasks sourced from state-of-the-art machine learning papers. These tasks span diverse domains, including language modeling, mathematics, bioinformatics, and time series forecasting. AIRS-Bench tasks assess agentic capabilities over the full research lifecycle—including idea generation, experiment analysis and iterative refinement—without providing baseline code. The AIRS-Bench task format is versatile, enabling easy integration of new tasks and rigorous comparison across different agentic frameworks. We establish baselines using frontier models paired with both sequential and parallel scaffolds. Our results show that agents exceed human SOTA in four tasks but fail to match it in sixteen others. Even when agents surpass human benchmarks, they do not reach the theoretical performance ceiling for the underlying tasks. These findings indicate that AIRS-Bench is far from saturated and offers substantial room for improvement. We open-source the AIRS-Bench task definitions and evaluation code to catalyze further development in autonomous scientific research.

Date: February 10, 2026

Correspondence: Despoina Magka at despoinam@meta.com, Yoram Bachrach at yorambac@meta.com, Jakob Foerster at jnf@meta.com

Code: https://github.com/facebookresearch/airs-bench

1 Introduction

From the rise of deep learning through to the current era of Large Language models (LLMs), the rapid progress in machine learning (ML) has been largely benchmark-driven, making robust evaluations a key requirement (Hardt, 2025). This trend is reinforced by the current *reviewing crisis*, which highlights the limits of human judgment in assessing the quality of research contributions (Xu et al., 2022; Lawrence, 2022). Recent improvements in LLM technology have extended their abilities from simple tasks to complex agentic workflows, including scientific reasoning and coding (Khatri et al., 2025; Andrews et al., 2025). Whilst early LLM research focused on improving the capabilities of LLMs through better data and pre-training, recent work has focused on extending their abilities by leveraging more test-time compute (Yao et al., 2023a), in particular by leveraging *scaffolds* that iteratively query the LLM at test-time so as to obtain a better response. Such scaffolds can enable a test-time search of the solution space by incorporating environment feedback (Nathani et al., 2025; Toledo et al., 2025) and yield much more performant solutions. Despite this potential, we lack a standardized framework to measure how well these agents perform the actual work of a research scientist. To fill this gap, we introduce AIRS-Bench.

We adopt the broadly accepted definition of an agent as a *computer system situated in an environment that*

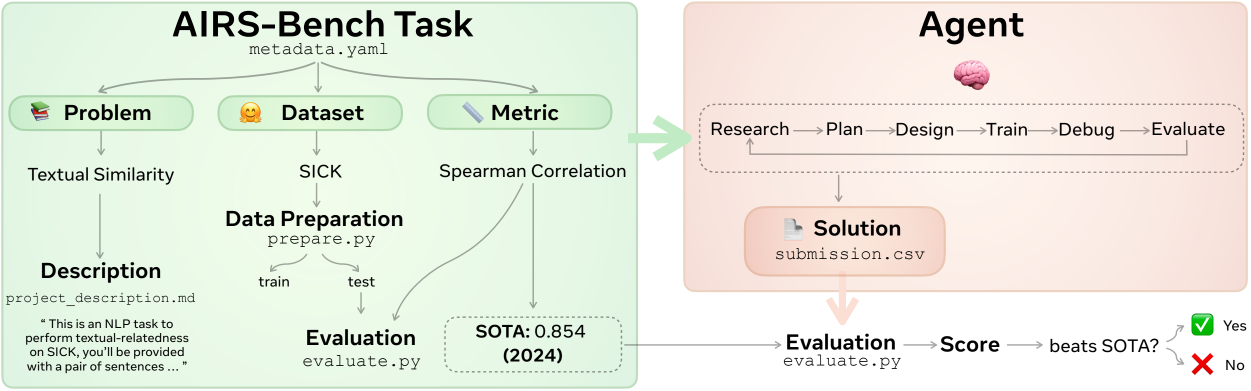


Figure 1 Example of an AIRS-Bench task. Each task is specified by a {problem, dataset, metric} triplet. The **problem** defines the core computational challenge to be solved (e.g. textual similarity); the **dataset** specifies which data to solve the challenge over (e.g. SICK); finally, the **metric** is used to quantify performance (e.g. Spearman correlation). The agent receives the full task specification and is expected to develop a solution that in most cases generates predictions on the test labels file, which are then evaluated and compared with the state-of-the-art result.

is capable of autonomous action in order to meet its design objectives (Wooldridge and Jennings, 1995). In our context, an *agent* is characterized as an *LLM* augmented by a *scaffold*. The LLM is the underlying probabilistic model that serves as the agent's primary reasoning core, either self-hosted OSS model or API-based. The scaffold acts as an orchestrating layer, providing the necessary mechanisms to translate LLM outputs into systematic exploration of the solution space. The scaffold is typically implemented by a *harness*, the execution framework responsible for instantiating and managing diverse scaffolding configurations. Examples are the ReAct scaffold (Yao et al., 2023b), implemented within the MLGym harness (Nathani et al., 2025) or the Monte Carlo Tree Search (MCTS) scaffold (Kocsis and Szepesvari, 2006), implemented within the AIRA-dojo harness (Toledo et al., 2025). We experiment with both sequential (linear) and parallel scaffolds. Sequential scaffolds follow a linear execution loop, where each LLM query is conditioned on feedback from previous actions (Yao et al., 2023b; Nathani et al., 2025). Parallel scaffolds, by contrast, maintain and grow a population of potential solutions, utilizing data structures such as trees to guide exploration (Novikov et al., 2025; Sharma, 2025; Toledo et al., 2025).

A growing area of research for LLM agents is represented by the automation of AI research itself. We call agents designed to automate and accelerate AI research *AI Research Agents*. Accurately evaluating the performance of such agents, however, remains a significant challenge. ML has long struggled with reproducibility and statistical noise (Recht et al., 2018; Hardt, 2025), and the agentic paradigm adds further complications:

- Data contamination: LLMs are trained on vast amounts of internet data, and they often memorize benchmark solutions. This makes it difficult to assess whether an agent is truly "reasoning" on a task.
- Environmental standardization: agentic environments are difficult to standardize across different studies, and it is often unclear if success comes from the agent's capabilities or the specific way the environment was built.
- Computational cost: The high computational overhead of each autonomous run makes it difficult to conduct the extensive trials necessary to obtain statistically significant results.

These factors have exacerbated the *evaluation crisis* of AI Research Agents, as performance on existing benchmarks is increasingly obscured by pretraining leakage, inconsistent environment setups, and noisy empirical evaluations (Haimes et al., 2024; Dehghani et al., 2021). We introduce the *AI Research Science Benchmark* (AIRS-Bench) to address these limitations and provide a standardized evaluation of AI Research Agents. The philosophy behind AIRS-Bench is that, much like the progress in ML, the advancement of AI Research Agents should be a benchmark-driven process. AIRS-Bench is a suite of 20 agentic tasks, curated from recent state-of-the-art (SOTA) literature. This ensures that tasks are both challenging and relevant to the broader ML community. In our design, agents are not required to tackle these tasks through direct inference. Instead, they must generate the code necessary to train and validate an ML model. We then evaluate the agent's capabilities by executing the code it generated and measuring the model performance. This approach effectively evaluates the agent's capacity to function as an autonomous research scientist.

To both standardize evaluations and make the tasks closely match their original versions in the ML literature, we developed a *task configuration standard* that encapsulates an AIRS-Bench task (see Figure 1 for an example) and a task creation pipeline based on semi-manual sourcing, creation, reviewing and verification of tasks. Our proposed AIRS-Bench task standard can be adapted and extended to virtually any ML problem, effectively democratizing AI agentic research. For fair evaluations across all agents we have designed our experiments to make runs as comparable as possible by factoring in infrastructure issues (e.g. crashing runs) and eliminating environment discrepancies. To investigate the impact of different scaffolds, we evaluate both the AIRA-dojo (Toledo et al., 2025) and MLGym (Nathani et al., 2025) harnesses combined with different LLMs (CWM (Carbonneaux et al., 2025), GPT-4o (OpenAI, 2024a), gpt-oss-20b and gpt-oss-120b (OpenAI, 2024b), o3-mini (OpenAI, 2024c), Devstral (Rastogi et al., 2025)). We also introduce an evaluation protocol, consisting of different metrics and aggregating results across seeds and tasks to allow for statistically robust and interpretable results.

The key contributions of the paper are:

- **Uncontaminated benchmark for AI Research Agents**: AIRS-Bench uniquely assesses agents on the end-to-end ML research workflow: idea generation, methodology design, experiment analysis, and iterative refinement—without access to baseline code. This enables a realistic evaluation of agentic research abilities. AIRS-Bench tasks cover a variety of machine learning research problems, ranging from NLP, math and code to biochemical modeling and time series forecasting. The benchmark is designed to be compatible with multiple agentic frameworks (i.e. harnesses), supporting robust and fair comparisons across different agent architectures and tool integrations.
- **Task configuration standard**: we introduce a task configuration standard and fixed evaluation metrics, to ensure reproducibility and minimize runtime and environment inconsistencies. We establish quality by leveraging human checks throughout task building, reviewing, and verification.
- **Empirical analysis**: we benchmark all tasks across frontier open and closed-source models and different scaffolds, revealing substantial variation in performance. We closely inspect cases where agents are shown to match or beat human SOTA and analyze the agent-generated solutions—including cases where agents discover effective SOTA-exceeding combinations of approaches.
- **Evaluation protocol**: we define a suite of metrics assessing different aspects of agents' capabilities including valid submission rates, normalized performance scores, and Elo ratings. Normalization enables us to aggregate performance across tasks by applying transforms that map 0.0 scores to the weakest valid solution and 1.0 to human SOTA. The metrics used are designed to accurately reflect progress on developing state-of-the-art AI Research Agents. We report results on these metrics across the range of selected agents; our results suggest that the benchmark is far from solved, leaving plenty of headroom for development of AI Research Agents.

The rest of the paper is organized as follows. Section 2 contains an overview of existing benchmarks within the AI research domain. Section 3 introduces the agent, scaffold and harness definitions and section 4 presents the benchmark structure and creation methodology. Sections 5 and 6 outline our experimental design and results. We conclude with discussing learnings and key findings in Section 7.

2 Related Work

We examined a number of benchmarks evaluating the ability of AI agents to carry out scientific research, including ML-related agentic tasks. Below, we outline several trends we identified and how they relate with AIRS-Bench.

Full cycle of Scientific Method. Many earlier papers create agentic tasks from available sources: Github repositories (CSR-bench; Xiao et al., 2025), top-tier conference papers and their data (PaperBench; Starace et al., 2025a), Kaggle competitions (MLE-bench; Chan et al., DSBench; Jing et al., 2025), ML research

Benchmark	Task Composition and Origin	AI Research Data	Task Horizon	Scientific Method				No Baseline	Agent Compute
				(H)	(I)	(E)	(A)		
AIRS-Bench (ours)	20 tasks from 17 machine learning papers with state-of-the-art results	✓	long	✓	✓	✓	✓	✓	high GPU
Automated LLM Speedrun (Zhao et al., 2025)	76 tasks from the NanoGPT Speedrun Github repo	✓	long	✗	✓	✓	✓	✓	high GPU
Auto-Bench (Chen et al., 2025a)	6 graph discovery tasks from chemistry and social networks	✗	short	✓	✗	✗	✗	✓	not specified
CORE-Bench (Siegel et al., 2024a)	270 tasks from 90 social, medical, and computer science papers	✓	medium	✗	✓	✓	✓	✗	low GPU
CSR-Bench (Xiao et al., 2025)	107 Github repos from CV, NLP and interdisciplinary ML papers	✓	not specified	✗	✓	✓	✓	✗	not specified
MLE-Bench (Chan et al., 2024)	75 Kaggle competitions	✗	long	✓	✓	✓	✓	✓	high GPU
MLGym-Bench (Nathani et al., 2025)	13 tasks from supervised learning, RL and algorithmic reasoning problems	✓	medium	✓	✓	✓	✓	✓	high GPU
ML-Agent-Bench (Tang et al., 2024)	13 tasks from Kaggle, computer science papers, and textbooks	✓	medium	✓	✓	✓	✓	✗	low GPU
SWE-Bench (Jimenez et al., 2023)	2,294 Github issues from open-source repos	✗	short	✗	✓	✓	✓	✓	CPU
SciReplicate-Bench (Xiang et al., 2025a)	100 tasks from 36 NLP papers	✓	medium	✗	✓	✓	✓	✗	not specified
PaperBench (Starace et al., 2025b)	8,316 rubrics from 20 ICML papers	✓	short	✗	✓	✓	✓	✗	low GPU
ResearchBench (Liu et al., 2025a)	Hypothesis generation task for 1386 papers across the sciences	✓	not specified	✓	✗	✗	✓	✓	not specified
RE-Bench (METR, 2024)	7 LLM pretraining/coding tasks and 71 human attempts	✗	medium	✓	✓	✓	✓	✗	high GPU
LMR-Bench (Yan et al., 2025)	28 code reproduction tasks from 23 NLP papers	✓	not specified	✗	✓	✗	✗	✗	not specified
PostTrainBench (Rank et al., 2025)	Post-train 4 base LLMs to maximise perf across 5 benchmarks	✗	medium	✓	✓	✓	✓	✗	high GPU

Table 1 Comparison of 14 popular agentic AI research benchmarks with AIRS-Bench across key evaluation dimensions. Task horizon refers to the time window provided to solve the problem and it can be short (<1 hour), medium (1-12 hours) or long (>12 hours). We indicate whether the benchmarks assess for the hypothesis generation (**H**), implementation (**I**), experimentation (**E**) and analysis (**A**) stages of the scientific pipeline. Compute refers to the resources provided to the agent and it can be CPU, low GPU ($\leq$ 1 hour per task) and high GPU (>1 hour per task). No baseline refers to whether the agent has access to a baseline solution to tackle the problem.

competitions (MLRC-Bench; Zhang et al., 2025), and carefully selected cross-domain papers (SciReplicate-bench; Xiang et al., 2025b). The experimental cycle of these benchmarks can include ideation, implementation, experimentation, analysis and comparison to previous results, with several benchmarks focusing on separate stages of the cycle, such as IdeaBench (Guo et al., 2024), LiveIdeaBench (Ruan et al., 2025), AI Idea Bench (Qiu et al., 2025) and ResearchBench (Liu et al., 2025b) for ideation, FML-bench (Zou et al., 2025) for idea novelty estimation, SurveyBench (Sun et al., 2025) for literature review, and DataGovBench (Liu et al., 2025c), DCA-Benchmark (Huang et al., 2025), DA-Code (Huang et al., 2024b), and DS-1000 (Lai et al., 2022) for data quality, implementation and analysis. AIRS-Bench, on the other hand, requires the agent to excel in every step of the scientific method to perform well on the benchmark tasks, spanning hypothesis generation, implementation, experimentation, and analysis.

Access to Baseline Solution. Benchmarks differ significantly in two ways: whether the agent is granted access to a baseline solution, and the degree of "saturation" within the underlying problem. AIRS-Bench prioritizes unsaturated tasks, and does not provide a starter solution to the agent. This approach makes AIRS-Bench tasks challenging, as agents must navigate a longer reasoning horizon to make progress independently.

Different environments. Recent benchmarks leverage a wide range of different environments. While most setups closely resemble AIRS-Bench—requiring agents to interact via prompts and code execution—others incorporate gamified environments (DiscoveryWorld; Jansen et al., 2024) or physics engines (FEABench; Mudur et al., 2024). Task sets are typically manually curated, though some benchmarks (DiscoveryBench; Majumder et al., 2024, SUPER; Bogin et al., 2024) also include synthetic tasks. Some papers simulate user interactions within the environment (AppWorld; Trivedi et al., 2024).

Diversity of domains. Beyond core ML tasks, there is a growing trend to expand agentic evaluation into broader scientific domains, such as bioinformatics, chemistry, and physics. We identify more than 20 different scientific domains covered across recent benchmarks, including DiscoveryWorld (Jansen et al., 2024), DiscoveryBench (Majumder et al., 2024), CURIE (Cui et al., 2025), FEABench (Mudur et al., 2024), ScienceAgentBench (Chen et al., 2025b), CORE-bench (Siegel et al., 2024b), AUTObench (Chen et al., 2025a), ResearchBench (Liu et al., 2025b) and BioMLbench (Miller et al., 2025). AIRS-Bench features text and tabular-based tasks across seven categories of ML problems, including language modeling, mathematics, code generation, molecular modeling, and time-series forecasting. Currently, AIRS-Bench provides a comprehensive testbed for agents, with the potential for future expansion into additional scientific domains.

We selected a subset of benchmarks that most closely resemble AIRS-Bench—specifically those evaluating an agent's ability to conduct AI research independently. Table 1 summarizes this comparison across several dimensions: task composition and origin, reasoning horizon, the stages of the scientific method covered, access to baseline solutions, and compute requirements. AIRS-Bench is composed of tasks grounded in AI research data and crafted so that AI research agents (i) tackle long-horizon research challenges that mirror the complexity of scientific discovery (ii) engage with the complete scientific pipeline from hypothesis generation to implementation, experimentation and analysis in an iterative fashion (iii) operate without a starting solution to ensure unbiased evaluation and promote original research and (iv) leverage substantial computational resources to enable thorough solution space exploration and tackle modern AI research problems. These characteristics make AIRS-Bench well-suited for tracking progress toward developing state-of-the-art AI Research Agents.

3 Agents, Scaffolds, Harnesses

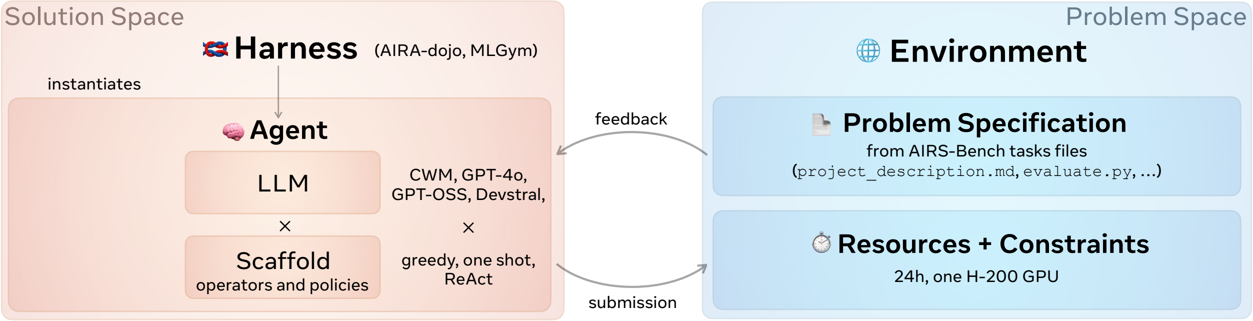


Figure 2 We define an **agent** as a pair consisting of a large language model (LLM) and a scaffold. A **scaffold** comprises a set of mechanisms, such as operators and search algorithms, that enable the LLM to explore the solution space effectively. Scaffolds are instantiated by a **harness**, which serves as a system that encapsulates the agent and manages its research process. The **environment** provides the agent with the problem specifications, as well as any constraints and resources available for its exploration.

In line with the broader agent research literature, we view an agent as a computer system that is situated in some environment and and is able to act autonomously in this environment in order to achieve its design objectives (Wooldridge and Jennings, 1995). For our specific context of LLM agents for AI Research, we define an *agent* as the combination of an LLM and a *scaffold*. A *scaffold* is an algorithm, expressed as a a set of operators and search policies governing how the agent searches the space of possible solutions for the task at hand. Scaffolds are instantiated within a *harness*—a system that wraps the agent and manages its execution (see Figure 2). Examples of harnesses include AIDE (weco.ai, 2024; Jiang et al., 2025), Claude

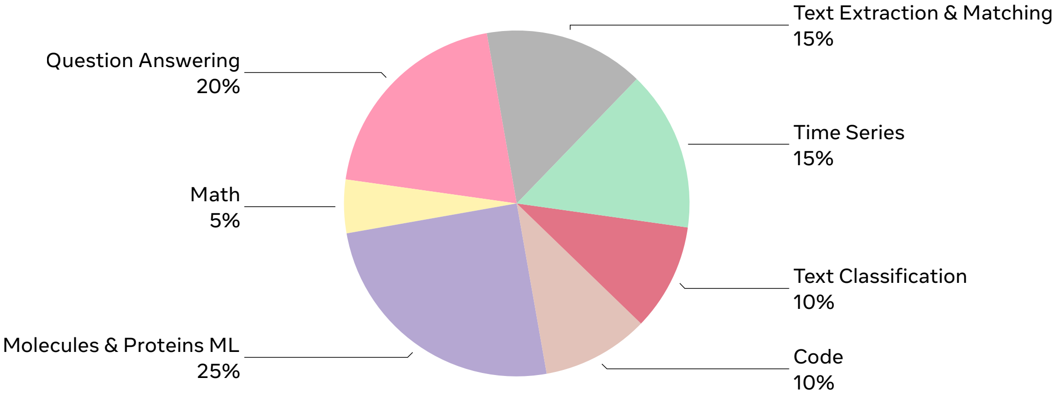


Figure 3 Distribution of AIRS-Bench tasks by category. We consider 7 distinct task categories in total: *Code*, *Math*, *Molecules & Proteins ML*, *Question Answering*, *Text Classification*, *Text Extraction & Matching*, and *Time Series*.

Code, [1] OpenCode, [2] MLGym (Nathani et al., 2025) and AIRA-dojo (Toledo et al., 2025). For instance, we refer to greedy search as a scaffold that is instantiated within the AIRA-dojo harness.

AIRA-dojo is a harness that enables the agent to evolve its solution through a set of operators and a search policy. The search policy (e.g., greedy search, Monte Carlo Tree Search (Kocsis and Szepesvari, 2006), evolutionary algorithms (Novikov et al., 2025; Sharma, 2025; Romera-Paredes et al., 2023)) guides the exploration of the solution space, while the operators modify existing solutions to generate new candidate solutions. This process results in a tree structure, where each node represents a Python code solution, created by one of the following operators: (1) *Draft*, which generates the initial set of solutions; (2) *Debug*, which identifies and corrects errors within a given node; and (3) *Improve*, which enhances a solution to increase its performance according to specified evaluation criteria. AIRA-dojo operators enhance AIDE to (i) promote solution diversity, by inserting into the context solutions of sibling nodes, and (ii) facilitate bug fixing, by including in the context the entire ancestral memory of the solution's debug chain. AIRA-dojo allows access to the internet, including additional data and pretrained models, which are loaded in the cache for the agent to use according to prompting instructions. AIRA-dojo's operator prompts can be found in Appendix C.2.

MLGym is a harness through which an agent can sequentially improve its initial solution in a ReAct-like manner (Yao et al., 2023b). The agent can develop a solution based on its own ideation and feedback from the execution of the implementation. Access to the internet is also allowed in a manner similar to AIRA-dojo, with the option to cache pretrained models and instruct the agent about them in the prompt. The agent has access to tools and bash, and the full system prompt is present in Appendix C.1.

4 Method

AIRS-Bench consists of 20 tasks extracted from 17 machine learning papers and 16 different datasets. To ensure selection of highly impactful tasks for the community, we sourced tasks and SOTA results from papers published at well-known AI conferences and journals and arXiv preprints (see Appendix F for a distribution of publication venues and years). The initial list of tasks was sourced from PapersWithCode's leaderboards (Taylor, 2020; Kardas et al., 2020) and filtered down for datasets that had a state-of-the-art result published between 2020 and 2025; additional criteria applied at this stage included availability of the dataset (preferably via HuggingFace) and existence of a train and a test split. After verifying these requirements, we manually created each task and reviewed the SOTA result associated with it, by confirming the number reported in the cited paper, checking that the evaluation metric and dataset splits were identical and updating the SOTA result if a better score could be found. By following this methodology, we initially created and evaluated approximately 100 tasks from ∼85 different machine learning papers and datasets. We subsequently selected 20 out of these to ensure tractability and accuracy of the benchmark. The process is outlined in Appendix A.

[1] https://code.claude.com/docs/
[2] https://github.com/anomalyco/opencode

Field	Value
name	MathQuestionAnsweringSVAMPAccuracy
research_problem	Math Question Answering
dataset	ChilleD/SVAMP
metric	Accuracy
metric_lower_is_better	false
config	default
train_split	train
test_split	test
input_columns	question_concat
scoring_column	Answer
category	Math
sota_score	0.942
sota_year	2026
sota_venue	Frontiers of Computer Science
sota_paper_title	*Achieving >97% on GSM8K: Deeply Understanding the Problems Makes LLMs Better Solvers for Math Word Problems*
sota_paper_url	https://arxiv.org/pdf/2404.14963v5
dataset_paper_url	https://arxiv.org/abs/2103.07191

Table 2 Core fields of the `MathQuestionAnsweringSVAMPAccuracy` task stored in its `metadata.yaml` file.

4.1 Tasks Diversity

Figure 3 show the 7 distinct categories that AIRS-Bench's 20 tasks are organized into, with the tasks spanning a broad spectrum of ML problems. The largest share is dedicated to the NLP domain, comprising *Question Answering* (4/20, open-ended and extractive question answering over diverse contexts), *Text Extraction & Matching* (3/20, e.g. coreference resolution, semantic similarity) and *Text Classification* (2/20, sentiment analysis, document categorization) tasks. Then, *Molecules and Proteins ML* tasks (5/20) focus on predicting molecular properties and solving biochemical modeling problems, whereas *Time Series* tasks (3/20) perform forecasting over temporal data. Finally, *Code* (2/20) and *Math* (1/20) tasks correspond to code generation/retrieval and mathematical reasoning, respectively.

The datasets included in AIRS-Bench span both unstructured data (plain text) and structured modalities (such as tables and graphs), presenting challenges for even the most advanced models (Fatemi et al., 2023; Lupidi et al., 2025).

4.2 Key Task Fields

Table 2 lists the core task fields of an AIRS-Bench task. Specifically, `name` contains the unique task name, `research_problem` describes the problem the agent is asked to solve, `dataset` contains the HuggingFace dataset identifier, `config` corresponds to the subset of the HuggingFace dataset used and `train_split` and `test_split` to the HuggingFace datasets splits used by the agent for training and evaluation, respectively. The `metric` field contains the metric the agent is asked to minimize or maximize (depending on the value of `metric_lower_is_better`), while `input_columns` contains the dataset columns the agent can utilize to solve the research problem and `scoring_column` is the column the agent's solution will be evaluated against. Finally, `category` describes the domain of the research problem, `sota_paper_title`, `sota_paper_url`, `sota_year` and `sota_venue` hold information related to the paper containing the state-of-the-art result and `sota_score` is the state-of-the-art value of the `metric` appearing in the paper; the agent is not accessing any of the information stored in the category and sota-related fields.

4.3 Task Files

Table 2 describes most of the key information that forms an AIRS-Bench task. This information requires additional code and data to be run on the AIRA-dojo and MLGym scaffolds. Sections C.1 and C.2 of the

Appendix contain the system prompts of the two scaffolds. The full task specification includes a folder, whose name is the same as the task name, and a number of files that are required for the agent to solve the task within AIRA-dojo. The task files are organized in such a way that they can be easily and programmatically converted into files required by different harnesses; we show this by converting them into task definition files for the MLGym agentic framework. The linked Github repository[3] contains the AIRS-Bench task specifications for AIRA-dojo and MLGym along with scripts for AIRA-dojo-to-MLGym format conversion and scripts for preparing the experimental environment (e.g. dataset downloads). In the remaining of this section, we describe the format and purpose of the task definition files for AIRA-dojo.

4.3.1 project_description.md

The `project_description.md` file contains the instructions provided to the agent to complete the task in the form of a lengthy and appropriately structured prompt; see Appendix B.1.1 for a full example. The prompt is divided into three sections: a description of the research problem, a description of the dataset and an explanation of the evaluation setup.

The research problem is presented in a sentence that describes the objective of the problem along with the column of the dataset that the predictions will be evaluated against. For the running example, this is:
“Your task is to solve math word problems. Each example presents a short story followed by a specific question. Your task is to read the text and predict the correct numerical answer. Your predictions will be scored against the `Answer` column of the test set ”

In the dataset description, we report the structure of the HuggingFace dataset. In particular, we specify which repo the data comes from and the dataset schema (features) with an overview of the columns and their datatypes. This helps the agent understand how the data looks like even if the scaffold does not provide a lookahead function. All the data used by the agent during a task is pre-downloaded and exported to the agent’s container.

Lastly, we explain to the agent how to submit the solution and how it will be scored: the agent is expected to submit its solution in the form of a `.csv` file containing the predictions on the test split, which has the benefit of a standard output format. We also provide the agent with the code of the evaluation script (`evaluate.py`, explained below) that contains the metric implementation and will be used to score the agent-produced `submission.csv` file against the test data.

4.3.2 prepare.py and evaluate_prepare.py

The `prepare.py` and `evaluate_prepare.py` files contain the one-time data preparation logic for the agent to solve the problem and for the evaluation of the agent’s solution, respectively. Please note that there are differences between the two settings, as the test labels need to be removed while the agent is building its solution; the two scripts take care of these requirements.

4.3.3 evaluate.py

The `evaluate.py` file is the evaluation script used to score the agent’s submissions against the test data. See Appendix B.1.2 for the evaluation script of the `MathQuestionAnsweringSVAMPAccuracy` task. The script contains three core functions: `load_test_set` where the script loads the test data, `evaluate` which implements the metric used to score the submissions, and `cli` which orchestrates loading of the agent’s submissions and test data, running the `evaluate` method on these, and reporting the results to stdout.

4.3.4 metadata.yaml

The `metadata.yaml` file contains all the metadata about the task (same as the fields of Table 2 described in detail above) along with additional requirements to run the task (like libraries used by the evaluation script that need to be installed). See Appendix B.1.3 for the `metadata.yaml` file of the `MathQuestionAnsweringSVAMPAccuracy` task.

[3] https://github.com/facebookresearch/airs-bench

4.3.5 utils.py

The `utils.py` file is an optional file to consolidate overlapping code between the `prepare.py`, `evaluate.py` and `evaluate_prepare.py` files. Examples include normalization transforms used for data preparation or bespoke label extraction logic.

4.3.6 Train and test datasets

All datasets have been downloaded beforehand using the `prepare_hf_datasets_text.py` script. The data required for the task is then mounted to the agent's container and prepared using the code within `prepare.py` and `evaluate_prepare.py`. The data folder always contains a train split under a folder whose name is the value of the `train_split` field of `metadata.yaml` and a test split under a folder similarly named using the `test_split` field. For the train split, one preparation step is removing all but the relevant columns for the task (and transforming their content if needed). This is to ensure that the model does not have access to extra data that might hint to the solution. The test split contains the test labels when used by the evaluation script to score the agent's submissions, but it does not contain the labels when accessed by the agent to look at the structure of the test set and solve the problem.

5 Experiments

5.1 Evaluation Setup

We evaluate AIRS-Bench using two harnesses, MLGym and AIRA-dojo. To isolate the effects of design choices associated with different harnesses, we ensured similar constraints and resources across all runs. For further details, refer to Table 7 in Appendix C. The greedy scaffold within AIRA-dojo explores several solutions through a tree-based search policy, while MLGym operates sequentially within one reasoning stream. Each run lasts for 24 hours with access to one H-200 GPU. We launch each run of each task at least 10 times (which we refer to as 10 "seeds").

Throughout AIRS-Bench evaluations with MLGym and AIRA-dojo harnesses, agents are allowed to access HuggingFace checkpoints, permitting the use of pretrained models. To facilitate this process and to mitigate HuggingFace rate limits, we locally cache a number of pretrained checkpoints. Note that the cache does not offer access to the latest foundational models and the most recent cached model dates back to 2021. The full list can be found in section E of the Appendix. For both AIRA-dojo and MLGym runs, agents were explicitly instructed about the existence of the cache. Across all experiments, we do not provide agents with any information regarding the methodology or score of the SOTA paper. We hypothesize that some of the tasks in the benchmark would have benefited from more compute or time, but we kept costraints uniform across tasks to provide the agents with the same resources and push the limits of their ideation capabilities.

5.2 Metrics and Score Aggregation

We evaluated the performance of the agents on AIRS-Bench using three metrics: mean valid submission rate, average normalized score and Elo rating. The definitions of these metrics are provided below. Throughout all metrics and empirical results, we follow the terminology introduced in Section 3, for which an agent a is defined as the combination of a scaffold (e.g. Greedy) and a base LLM (e.g. gpt-oss).

For each task, the agents faced the challenge of being able to *submit* a valid solution, i.e. one that meets the requirements specified in the task description and yields a valid score. Our first evaluation metric is thus the mean **valid submission rate** (VSR) across tasks for an agent a, defined as

$$\overline{\text{VSR}}_a = \frac{1}{N_a} \sum_{t=1}^{N_a} \frac{valid_{a,t}}{total_{a,t}} \tag{1}$$

where $valid_{a,t}$ is the number of valid (successful) runs for agent a on task t, $total_{a,t}$ is the number of total runs for agent a on task t, N_a is the number of tasks over which agent a is being evaluated (i.e. the AIRS-Bench tasks) and agent a is a combination of a base LLM with a scaffold. The mean valid submission rate assesses the agents' capability to come up with a working solution and submit it confidently.

Producing an aggregate score for AIRS-Bench is challenging due to the high diversity of tasks included: most tasks have unique metrics, and even for tasks sharing the same metric (e.g. accuracy), ranges reported in the literature for each of them may vary significantly. To aggregate heterogeneous metrics and ranges into a common scoring system, we define the **normalized score** (NS) of an agent a on a task t as:

$$\mathrm{NS}_t^a = \frac{\phi_t(s_t^a) - \phi_t(s_t^{\min})}{\phi_t(s_t^{\mathrm{sota}}) - \phi_t(s_t^{\min})} \tag{2}$$

where $s_t^{\min}$ corresponds to the worst score observed across all seeds and all agents on task t, s_t^{sota} is the SOTA score on task t sourced from literature, s_t^a is the score achieved by agent a on task t and ϕ_t is a non-linear transformation. Note that $\phi_t(s_t^{\min})$ and $\phi_t(s_t^{\mathrm{sota}})$ will always correspond to normalized scores of 0 and 1, respectively. Equation 2 involves a two-step normalization: first, for a given task t, we apply a monotonic map ϕ_t onto the raw score s achieved by the agent's submission; second, we restrict the resulting scores within the $[0, 1]$ interval if the agent performs worse than SOTA and > 1 if the agent exceeds SOTA to enable subsequent aggregation across tasks. Specifically, we employ the **march of 9s** transform[4] as our choice of ϕ_t, defined as

$$\phi_t(s) = -\log_{10}(|s - s_t^{\mathrm{opt}}|) \tag{3}$$

where s_t^{opt} is the overall possible optimal score for the task (e.g., 1.0 for classification accuracy, 0.0 for regression error), as opposed to the best score obtained or SOTA (which e.g. for accuracy would be less than 1.0). This choice of ϕ_t is to adjust changes of the score so that they reflect intuitive measures of progress on the benchmark: this approach treats closing e.g. the gap from 0.99 to 0.999 as significant as closing the gap from 0.9 to 0.99, since both represent a tenfold reduction in the distance to optimal.[5] When averaging normalized scores across seeds, we include both failed (i.e., the agent fails to submit a valid solution) and invalid submissions (i.e., the agent submits a solution that does not yield a numerical score) treating them as submissions with 0 normalized score.

Lastly, to quantify the relative skill level of each agent evaluated, we employ the **Elo rating system** (Elo, 1967). We do so by treating each agent as a player. For each AIRS-Bench task, we treat each pairwise comparison of the agents' scores as a game, with the agent producing a better score winning that game. If two agents do not produce a valid submission or both produce the same score, that game is considered a tie.

We estimate the agents' ratings by fitting a Bradley–Terry (BT) model to the head-to-head outcomes, following the approach used in Chatbot Arena (Chiang et al., 2024). The model infers latent skill parameters (θ_a for agent a) such that the probability of one agent a outperforming another agent b follows a logistic function of their skill difference:

$$P(a > b) = \frac{1}{1 + \exp(\theta_b - \theta_a)} \tag{4}$$

We then convert the estimated skill parameters θ_a into Elo ratings using the following transformation, where N denotes the total number of evaluated agents:

$$R_a = \frac{400}{\ln(10)} \cdot \left[\theta_a - \frac{1}{N}\sum_{k=1}^{N} \theta_k\right] + 1000 \tag{5}$$

Unlike the classical Elo calculation, the BT model is order-invariant, making it well-suited for batch evaluations across multiple agents and tasks. Please note that in our setting we treat the human SOTA scores as an additional agent, the "SOTA" agent.

[4] The "march of nines" is a metaphorical expression, popularized by AI researcher Andrej Karpathy, to describe the vast and non-linear engineering effort required to achieve higher levels of reliability in AI systems (Karpathy and Patel, 2025).

[5] The reader can contrast this with a simpler but less representative transform definition, such as the identity transform, for which we present results in Section B of the Appendix

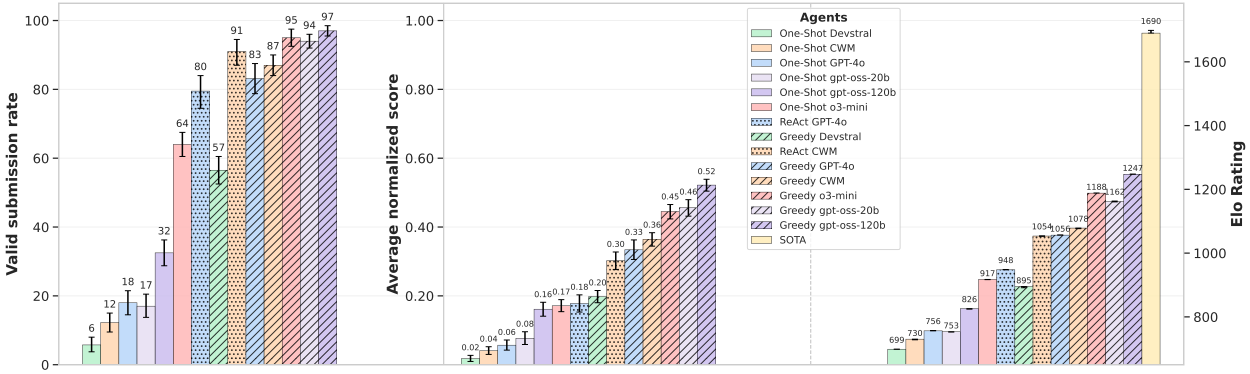


Figure 4 Overall performance of the 14 evaluated agents on the three metrics introduced in Section 5.2, namely valid submission rate, average normalized score and Elo rating. Results are ordered by increasing average normalized score.

6 Results

We evaluate a total of 14 agents, i.e. LLM-scaffold pairs. The language models used are the Code World Model (CWM), o3-mini, gpt-oss-20b and gpt-oss-120b, GPT-4o and Devstral-Small 24B. We evaluate three scaffolds: (i) One-Shot, where the agent can attempt solving the problem only once with the same set of AIRA-DOJO operators (by definition that would be the *Draft* operator only) (ii) Greedy, where the agent performs greedy search with the AIRA-DOJO operator set and (iii) ReAct, where the ReAct prompting technique implemented by MLGYM is powering the agent.

6.1 Comparing performance across agents

Figure 4 provides an overview of the three aggregate benchmark metrics introduced above. We observe that reasoning models (e.g. gpt-oss-120b, o3-mini) perform better in both one-shot and greedy settings with model size also affecting performance, i.e. gpt-oss-120b outperforms gpt-oss-20b by a significant margin. Moreover, tree-search methods benefit agents powered by both open-source and closed-source models, as suggested by e.g. the sizeable gaps between the performances of Greedy CWM and One-Shot CWM as well as Greedy GPT-4o and One-Shot GPT-4o agents. At the same time, performance of agents backed by linear scaffolds, such as ReAct CWM and ReAct GPT4o, stands in the middle. We observe that the relative ranking of agents shows similar trends for all three performance metrics; the ability to submit a valid solution correlates with the ability to submit performant solutions and the Elo ranking. We also notice that models such as o3-mini demonstrate high participation but also high loss rates, suggesting a tendency to submit more frequently but with less selectivity; in contrast, models like CWM participate less often but with higher confidence, reflecting distinct agent "personalities" and risk profiles. Finally, the majority of agent-task combinations achieve results between the o3-mini baseline and human SOTA, with a small but notable fraction (1.55%) exceeding SOTA, primarily driven by greedy search strategies.

We present valid submission rates for each agent in Figure 5, highlighting different ranges of valid submission rate across all tasks. We consider four submission rate ranges: *invalid* indicates that the agent failed to submit a valid solution across all its seeds; *low*/*medium*/*migh* correspond to the 1–33%, 34–66% and 67–100% of valid submission rates. Agents Greedy gpt-oss-120b and Greedy gpt-oss-20b lead with the smallest fractions of tasks yielding an invalid submission, at 6% and 7% respectively.

A breakdown of each agent's average normalized score per task is provided in Figure 6. For each agent, we report the percentage of tasks for which the agent yields one of five possible outcomes: (i) *invalid*, the agent does not submit a solution at all; (ii) *worst*, the agent produces the lowest score among all agents for that task; (iii) *below average*, the agents achieves a score below the average across all agents for that task; (iv) *above average*, the agent achieves a score above the average but is not the best; and (v) *best*, the agent achieves the highest score among all agents for that task. This breakdown highlights the distribution of each agent's performance across the benchmark tasks. Scores are normalized according to Equation 2, where ϕ_t is the

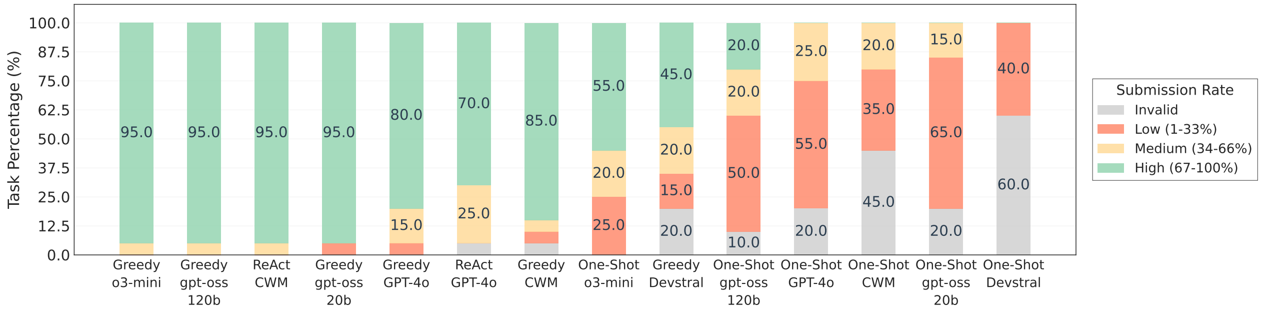


Figure 5 Submission rate distribution for the 14 agents tested. Each bar shows the distribution of submission rates across tasks for a given agent. The categories are defined as follows: *invalid* indicates that the agent did not provide any valid submission for that task (0% valid submissions); *low (1–33%)* indicates a valid submission for between 1% and 33% of seeds; *medium (34–66%)* indicates a valid submission for between 34% and 66% of seeds; and *high (67–100%)* indicates a valid submission for more than 66% of seeds. Agents are sorted by the combined percentage of seeds in the *medium* and *high* categories, highlighting those most reliable across the benchmark.

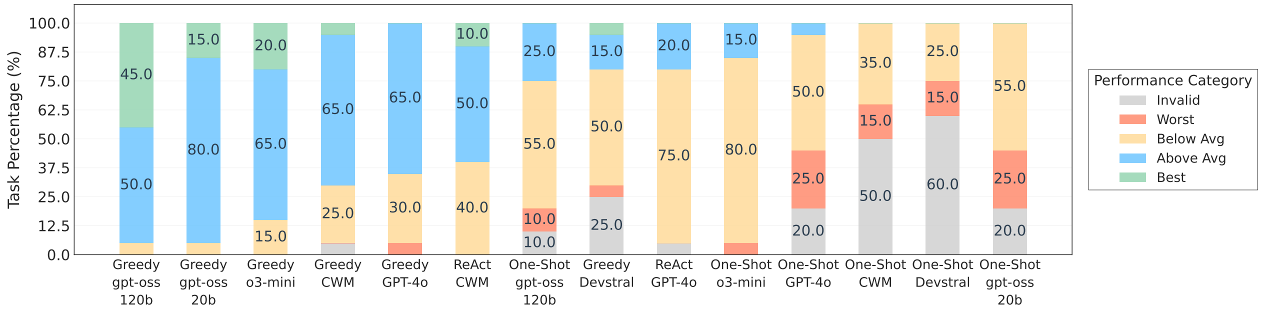


Figure 6 Performance distribution for the 14 agents evaluated. Each bar represents the percentage of tasks across all seeds for which a given agent falls into one of five performance categories: *invalid* (no valid submission for the task), *worst* (the lowest normalized score among all agents for the task), *below average* (normalized score below the mean but not the worst), *above average* (normalized score above the mean but not the best), and *best* (the highest normalized score for the task). Normalized scores are computed per task according to equations 2 and 3. Agents are sorted by the number of tasks for which they achieved the *best* and *above average* performances, highlighting those with the most consistent top performance across the benchmark.

march of 9s transform.

We report the mean valid submission rate defined in Equation 1 across the AIRS-Bench tasks in Figure. 7. On average, only 59.3% of the total submissions are considered valid, suggesting that even submitting a valid solution stretches the capabilities of the agents. We also observe that reasoning models and the greedy scaffolds offer an advantage, as performance of these agents is superior.

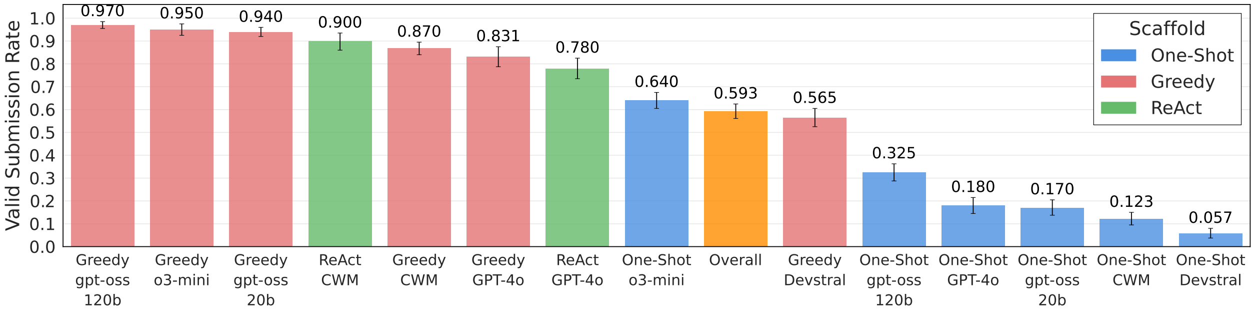


Figure 7 Mean valid submission rate (VSR) for the 14 agents evaluated, with error bars indicating the 95% confidence intervals. VSR is computed according to Eq. 1. The overall VSR across all runs and agents averages at 59.3% indicating that even submitting a valid solution is non-trivial for the agents' capabilities.

In Figure 8 we report the average normalized scores according to Equations 2 and 3. A more detailed

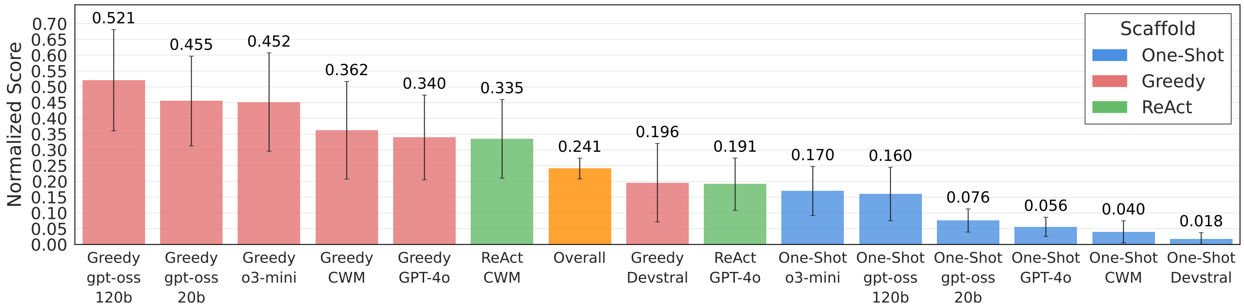


Figure 8 Average normalized scores for the 14 agents evaluated, with error bars indicating the 95% confidence intervals. Scores are computed according to Equations 2 and 3. The overall average normalized score across all runs and agents averages at 24.1%, highlighting the challenging nature of AIRS-Bench.

breakdown of the average scores per task is reported in Figure 9 with ϕ_t specified by Equation 3. The scores distribution in the figure reiterates the value of the AIRA-dojo harness in supporting agents to develop better solutions, with Greedy scaffolds (in red) distributing closer to SOTA than One-Shot ones (in blue).

Figure 9 depicts normalized scores computed according to Equations 2 and 3: each row corresponds to a task and each point relates to the normalized score achieved by an agent on that task and averaged across multiple seeds. For each task we average the normalized score across all agents and seeds and we use this mean normalized score to sort tasks by difficulty level. We stack tasks from the easiest to the most difficult going from top to bottom. The mapping between task numbers appearing in the y axis of Figure 9 and task names can be found in Table 6 of Appendix B. Tasks with points to the right of the 1.0 line indicate that the average score of that agent on that task exceeds human SOTA, which is not necessarily the case for all tasks with at least one seed (i.e. agent run) exceeding human SOTA. Figure 12 in Appendix B similarly presents normalized scores across tasks and agents, but using in Equation 2 the identity transform from Equation 8 (in Appendix B) instead of the march of 9s transform from Equation 3.

Based on the task ranking from Figure 9, in Figure 10 we group the AIRS-Bench tasks into four groups, each containing 5 tasks, and corresponding to an increasing level of difficulty: *easy* (tasks 1 to 5), *medium* (tasks 6 to 10), *hard* (tasks 11 to 15) and *expert* (tasks 16 to 20). Here we are averaging scores across 5 tasks, i.e. each point is the average over seeds of all 5 tasks in that difficulty bucket. We also observe in Figure 10 that while the normalized scores are all low and somewhat similar on the expert tasks, for the easier problems (and especially those in the easiest bucket), we see high variability between the scores that the agents achieved. The correspondence between task numbers and names is the same as the one in Figure 9 and can be found in Table 6 of Appendix B.

Finally, Elo ratings including human SOTA as an additional opponent alongside our agents are reported in Figure 11. The sizeable gap between the rating of the human SOTA player and the top-performing agent indicates that even the best agent is significantly below SOTA and the benchmark is very far from saturated.

6.2 Task Inspection: Success in Beating SOTA

Among the runs shown in Figures 9 and 12, we found cases where the agent's performance, at least in some of the seeds, was higher than the reported human SOTA, i.e. had a normalized score that was greater than 1. Overall, we identified 4 tasks where our agents surpass SOTA performance, as summarized across Tables 3-5. We examined these cases in depth to better understand the solution produced by the agent, and how it manages to outperform the human SOTA. Below, we provide a breakdown for a notable case where an agent outperforms human SOTA with an original solution.

Greedy gpt-oss-120b on TextualClassificationSickAccuracy. This is a Natural Language Inference (NLI) problem, and it employs the SICK (Sentences Involving Compositional Knowledge) dataset (Marelli et al., 2014a). Given a pair of sentences, a premise and a hypothesis, the goal is to determine the relationship between them, including: *entailment* (the hypothesis is true given the premise), *contradiction* (the hypothesis is false given the premise); and *neutral* (no conclusion can be drawn on the hypothesis given the premise). The evaluation metric is accuracy.

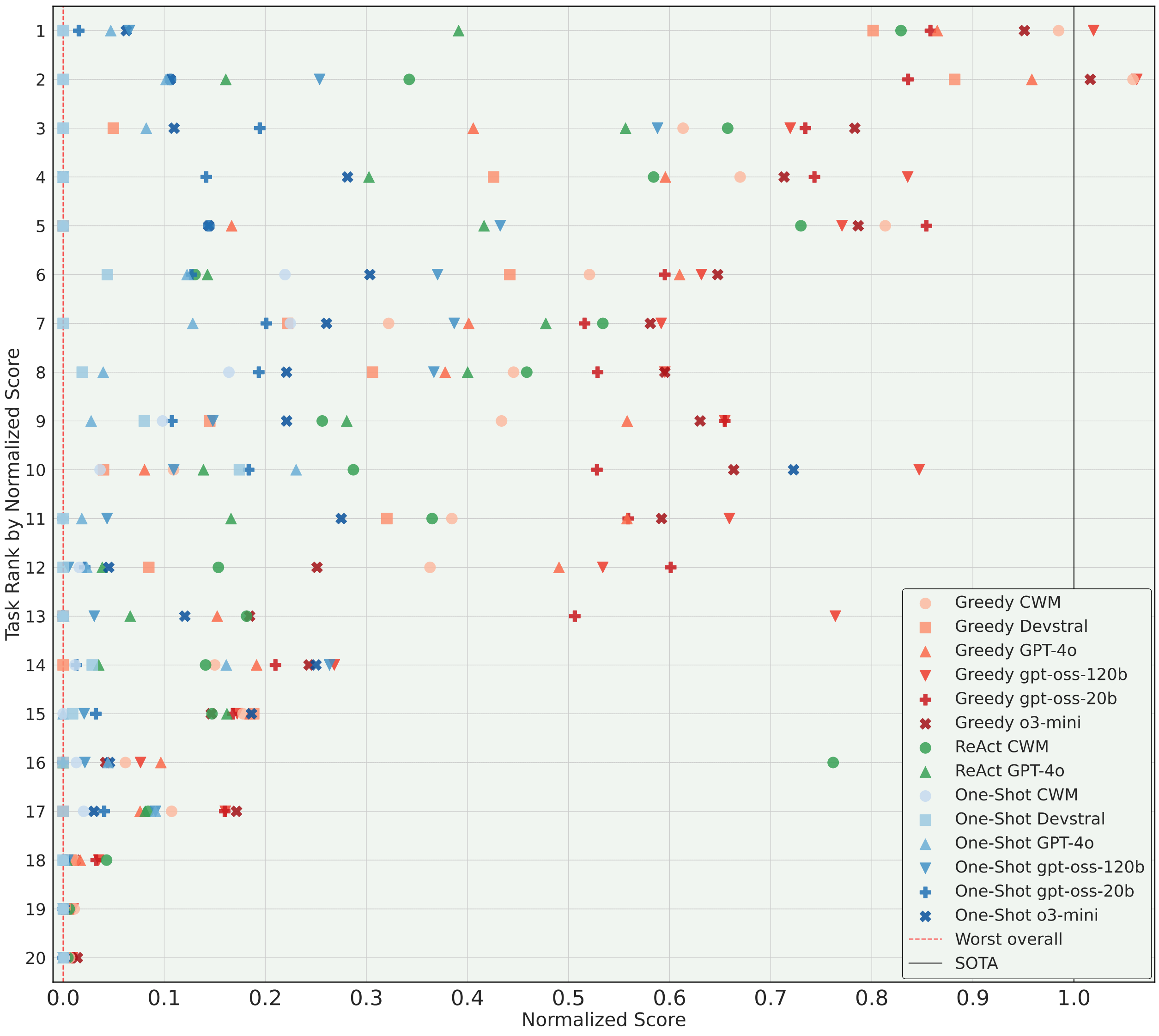


Figure 9 Average normalized scores with each row corresponding to an AIRS-Bench task and each point to an agent's normalized score for that task averaged across multiple seeds. For each task, the outcome of the worst-performing run is used as the baseline score. SOTA always corresponds to a normalized score of 1. Tasks are ranked in decreasing order according to the average score across all agents. See Table 6 for the correspondence between tasks numbers on the y axis and names.

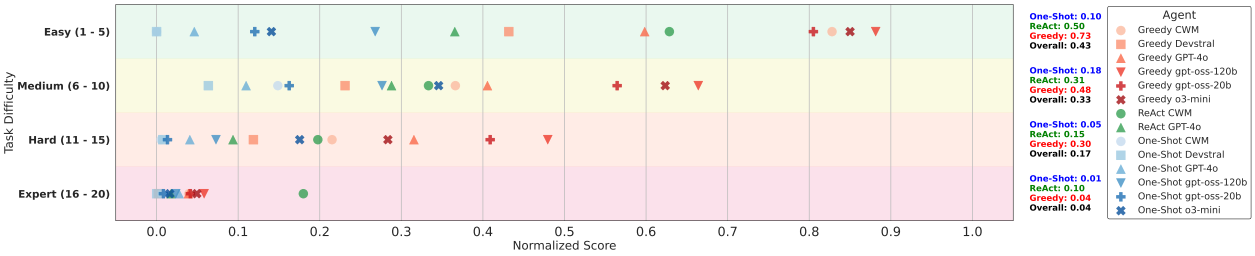


Figure 10 Normalized score per task difficulty level computed according to Equations 2-3. We divide the task ranking of Figure 9 into four categories with decreasing normalized scores: *easy*, *medium*, *hard* and *expert*.

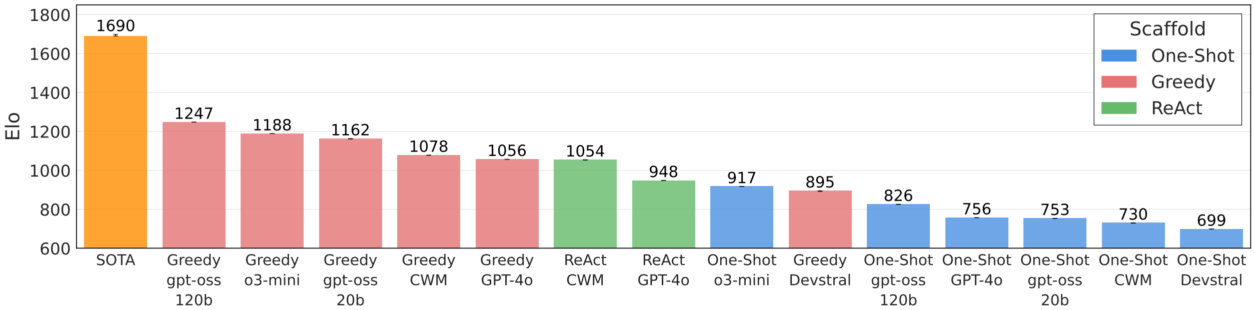


Figure 11 Elo ratings of all agents, estimated by fitting a Bradley–Terry model on the pairwise comparisons of agents' scores for each task. The human SOTA score is also included as an additional opponent. The Greedy scaffold outperforms other scaffolds in most cases. Bar height represents the median of a bootstrap distribution using 100 resamples, with the error bars representing the 95% confidence intervals.

The SOTA solution (Kalouli et al., 2023) is achieved by fine-tuning RoBERTa (Liu et al., 2019) on the original SICK training set and testing on the original SICK test set. The approach is straightforward: a single transformer model is fine-tuned on a specific training set, yielding a test accuracy of 90.5%.

The Greedy gpt-oss-120b agent, on the other hand, produces a two-level stacked ensemble that combines multiple transformer models and a meta-learner. RoBERTa-large and DeBERTa-v3-large (He et al., 2023), are independently fine-tuned on the SICK training set. Each model processes sentence pairs and outputs logits for each class. The training is performed using 5-fold stratified cross-validation, ensuring robust out-of-fold (OOF) predictions and preventing overfitting. The logits from both base models are concatenated to form a feature vector for each example. These stacked features are then used to train a logistic regression meta-learner, which learns to optimally combine the predictions of the two base models. During cross-validation, the meta-learner is trained on OOF logits, ensuring that the meta-features are unbiased and not overfitted. At test time, the base models are retrained on each fold, their test logits are averaged, and the meta-learner uses these combined logits to make the final prediction. This architecture leverages the complementary information captured by RoBERTa and DeBERTa, and the meta-learner can exploit patterns that may not be apparent to either base model alone, achieving a test accuracy of 93.1%.

Task	Score	Method
`TextualClassificationSick-Accuracy`	• **SOTA**: 0.90 • **Agent**: 0.93	• **SOTA** (Kalouli et al., 2023): Vanilla fine-tuning of RoBERTa. • **Agent**: Finetuned RoBERTa-large and DeBERTa-v3-large base models using stratified cross-validation. Out-of-fold logits from both models employed to train a logistic-regression meta-learner that combines base models' logits. Finally, base models retrained on all folds and meta-learner was used to produce final predictions.
`TextualSimilaritySick-SpearmanCorrelation`	• **SOTA**: 0.85 • **Agent**: 0.89	• **SOTA** (Huang et al., 2024a): Finetuned RoBERTa-large and novel loss function (CoSENT). • **Agent**: RoBERTa-base and RoBERTa-large finetuned to predict similarity scores. Produces similarity scores using cosine similarity of frozen Sentence-BERT. Used cross-validation to learn weights for averaging the similarity scores produced by all three models.

Table 3 AIRS-Bench tasks where the **Greedy gpt-oss-120b** agent surpassed human SOTA performances in at least one run, and achieves the best overall score. The left column displays the name of the task t; the middle column shows SOTA and the raw agent scores s_t^{sota} and s_t^a, respectively; the right column briefly summarises and compares the SOTA and the Agent solutions.

Task	Score	Method
CoreferenceResolution-WinograndeAccuracy	- **SOTA**: 0.85 - **Agent**: 0.88	- **SOTA** (Lin et al., 2020): Fine-tune T5-3B in a text-to-text setup, scoring answer options by output token probabilities ("entailment" vs. "contradiction"). - **Agent**: Vanilla finetuning of DeBERTa-v3-large with classifier head.

Table 4 AIRS-Bench tasks where the **Greedy gpt-oss-20b** agent surpassed human SOTA performances in at least one run, and achieves the best overall score. The left column displays the name of the task t; the middle column shows SOTA and the raw agent scores s_t^{sota} and s_t^a, respectively; the right column briefly summarises and compares the SOTA and the Agent solutions.

Task	Score	Method
TimeSeriesForecasting-RideshareMAE	- **SOTA**: 1.185 - **Agent**: 1.153	- **SOTA** (Gong et al., 2025): General transformer-based time-series foundation model (not finetuned on this dataset). - **Agent**: Trains a Bi-directional GRU

Table 5 AIRS-Bench tasks where the **Greedy CWM** agent surpassed human SOTA performances in at least one run, and achieves the best overall score. The left column displays the name of the task t; the middle column shows SOTA and the raw agent scores s_t^{sota} and s_t^a, respectively; the right column briefly summarises and compares the SOTA and the Agent solutions.

7 Conclusion

We introduced AIRS-Bench, a benchmark designed to rigorously evaluate the autonomous research capabilities of LLM agents in machine learning. Our benchmark covers 20 diverse, non-contaminated tasks spanning multiple domains, and is specifically constructed to assess agents across the full research workflow—from ideation and methodology design to experimentation and iterative refinement—without access to baseline code. Our results indicate a high variability in task performance, depending on both the LLM that the agent uses, and the harness it is based on. For most tasks, even the best performing agent is still significantly behind the human SOTA, showing that the benchmark is far from saturated. For a few tasks, our top agents managed to identify a solution outperforming the human SOTA. It would be interesting to see how much AI research agents can push the state-of-the-art further.[6]

We gathered a number of useful takeaways during building of AIRS-Bench and evaluating a number of agents across its tasks:

- Gaps in community infrastructure: within the current state of AI research, the task of tracking up-to-date SOTA became more challenging than ever. Both the growing amounts of paper submissions,[7] the high compute cost of reproducing experiments on large models and the lack of unified platforms to represent results contribute to the situation. A new shared space with standardized format, updates, and machine-readable configurations for all published machine learning research is needed.
- Performance gaps: several main factors cause the performance of the agents to be lower than it could be. The combinations of base LLMs with different scaffolds can lead to 1) problems with formatting, specifically submitting the correct solution after the experiments, 2) problems with saving intermediate results, 3) performance deterioration on main capabilities due to the context overflow. Longer agentic traces also lead to increased probabilities of misaligned behaviours and accumulated issues around code edits and debugging, which make it difficult to adhere to the methodology of the ML experimentation.
- Human bottlenecks: for the work to be scaled across new domains and a bigger volume of tasks,

[6] Note that even in the cases where the agent outperforms human SOTA, it is still interesting to see how far ahead the agent's solution is over the human baseline. Hence, tasks where the agent normalized score exceeds 1.0 are still interesting benchmark tasks.

[7] https://forum.cspaper.org/topic/76/submission-tsunami-at-neurips-2025-is-peer-review-about-to-collapse/2

automatic task onboarding pipelines will be needed. Current human validation procedures prevent the expansion at scale.

- Role of restrictions: The ML benchmarks for AI Agents tend to be computationally costly, both on the training and inference side. Given the significant resource constraints faced in agentic evaluations—such as computational costs, time limits, and token usage—we acknowledge the possible role of restrictions in the obtained results. Benchmark methodology commonly faces the choice of either evaluating the systems in the very well-defined restricted conditions or lifting most of them and comparing the best obtained results. Although we adhere to the first choice for the sake of future extensive ablations, lifting certain restrictions could enable more flexible and efficient agent behaviors in the future.

Our results demonstrate that scaffold design significantly impacts agent solution quality, highlighting opportunities to improve performance through algorithms that better leverage test-time compute. We release AIRS-Bench to help identify performance gaps in AI research agents and catalyze the development of better methods for accelerating scientific progress. As agentic capabilities advance, continued benchmark development will be essential. We hope this benchmark fosters transparency, reproducibility, and rigorous, standardized evaluation of LLM agents in frontier research contexts.

References

Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Grégoire Mialon, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Thomas Scialom, Vladislav Vorotilov, Mengjue Wang, and Ian Yu. ARE: Scaling Up Agent Environments and Evaluations, 2025. URL https://arxiv.org/abs/2509.17158.

Nabiha Asghar. Yelp dataset challenge: Review rating prediction. *arXiv preprint arXiv:1605.05362*, 2016.

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, 2024. URL https://arxiv.org/abs/2409.07440.

Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, Felix Kreuk, Emily McMilin, Michel Meyer, Yuxiang Wei, David Zhang, et al. CWM: An open-weights LLM for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.

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 Mądry. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering.

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.

Tingting Chen, Srinivas Anumasa, Beibei Lin, Vedant Shah, Anirudh Goyal, and Dianbo Liu. Auto-Bench: An Automated Benchmark for Scientific Discovery in LLMs, 2025a. URL https://arxiv.org/abs/2502.15224.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan R Routledge, et al. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, 2021.

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, 2025b. URL https://arxiv.org/abs/2410.05080.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios N. Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: an open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Hao Cui, Zahra Shamsi, Gowoon Cheon, Xuejian Ma, Shutong Li, Maria Tikhanovskaya, Peter Norgaard, Nayantara Mudur, Martyna Plomecka, Paul Raccuglia, Yasaman Bahri, Victor V. Albert, Pranesh Srinivasan, Haining Pan, Philippe Faist, Brian Rohr, Ekin Dogus Cubuk, Muratahan Aykol, Amil Merchant, Michael J. Statt, Dan Morris, Drew Purves, Elise Kleeman, Ruth Alcantara, Matthew Abraham, Muqthar Mohammad, Ean Phing VanLee, Chenfei Jiang, Elizabeth Dorfman, Eun-Ah Kim, Michael P Brenner, Viren Jain, Sameera Ponda, and Subhashini Venugopalan. Curie: Evaluating llms on multitask scientific long context understanding and reasoning, 2025. URL https://arxiv.org/abs/2503.13517.

Mostafa Dehghani, Yi Tay, Alexey A. Gritsenko, Zhe Zhao, Neil Houlsby, Fernando Diaz, Donald Metzler, and Oriol Vinyals. The Benchmark Lottery, 2021. URL https://arxiv.org/abs/2107.07002.

Arpad E Elo. The proposed uscf rating system, its development, theory, and applications. *Chess life*, 22(8):242–247, 1967.

Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. ELI5: Long form question answering. *arXiv preprint arXiv:1907.09190*, 2019.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models, 2023. URL https://arxiv.org/abs/2310.04560.

Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*, 2021.

Peiliang Gong, Emadeldeen Eldele, Min Wu, Zhenghua Chen, Xiaoli Li, and Daoqiang Zhang. Bridging Distribution Gaps in Time Series Foundation Model Pretraining with Prototype-Guided Normalization. *arXiv preprint arXiv:2504.10900*, 2025.

Sikun Guo, Amir Hassan Shariatmadari, Guangzhi Xiong, Albert Huang, Eric Xie, Stefan Bekiranov, and Aidong Zhang. Ideabench: Benchmarking large language models for research idea generation, 2024. URL https://arxiv.org/abs/2411.02429.

Jacob Haimes, Cenny Wenner, Kunvar Thaman, Vassil Tashev, Clement Neo, Esben Kran, and Jason Schreiber. Benchmark inflation: Revealing llm performance gaps using retro-holdouts, 2024. URL https://arxiv.org/abs/2410.09247.

Moritz Hardt. The Emerging Science of Machine Learning Benchmarks. Online at https://mlbenchmarks.org, 2025. Manuscript.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2111.09543.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring Coding Challenge Competence With APPS. *NeurIPS*, 2021.

Benhao Huang, Yingzhuo Yu, Jin Huang, Xingjian Zhang, and Jiaqi Ma. Dca-bench: A benchmark for dataset curation agents, 2025. URL https://arxiv.org/abs/2406.07275.

Xiang Huang, Hao Peng, Dongcheng Zou, Zhiwei Liu, Jianxin Li, Kay Liu, Jia Wu, Jianlin Su, and Philip S Yu. CoSENT: consistent sentence embedding via similarity ranking. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 32:2800–2813, 2024a.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. Da-code: Agent data science code generation benchmark for large language models, 2024b. URL https://arxiv.org/abs/2410.07331.

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, 2024. URL https://arxiv.org/abs/2406.06769.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138.

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.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents from becoming data science experts?, 2025. URL https://arxiv.org/abs/2409.07703.

Aikaterini-Lida Kalouli, Hai Hu, Alexander F Webb, Lawrence S Moss, and Valeria De Paiva. Curing the SICK and other NLI maladies. *Computational Linguistics*, 49(1):199–243, 2023.

Marcin Kardas, Piotr Czapla, Pontus Stenetorp, Sebastian Ruder, Sebastian Riedel, Ross Taylor, and Robert Stojnic. Axcell: Automatic extraction of results from machine learning papers, 2020. URL https://arxiv.org/abs/2004.14356.

Andrej Karpathy and Dwarkesh Patel. Andrej karpathy — agi is still a decade away. The Dwarkesh Podcast, Oct 2025.

Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S. Dhillon, David Brandfonbrener, and Rishabh Agarwal. The Art of Scaling Reinforcement Learning Compute for LLMs, 2025. URL https://arxiv.org/abs/2510.13786.

Levente Kocsis and Csaba Szepesvari. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–203. Springer, 2006.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation, 2022. URL https://arxiv.org/abs/2211.11501.

Neil Lawrence. The neurips experiment. Online at https://inverseprobability.com/talks/notes/the-neurips-experiment-snsf.html, 2022. Article.

Sheng-Chieh Lin, Jheng-Hong Yang, Rodrigo Nogueira, Ming-Feng Tsai, Chuan-Ju Wang, and Jimmy Lin. Tttttackling winogrande schemas. *arXiv preprint arXiv:2003.08380*, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019. URL https://arxiv.org/abs/1907.11692.

Yujie Liu, Zonglin Yang, Tong Xie, Jinjie Ni, Ben Gao, Yuqiang Li, Shixiang Tang, Wanli Ouyang, Erik Cambria, and Dongzhan Zhou. ResearchBench: Benchmarking LLMs in Scientific Discovery via Inspiration-Based Task Decomposition. *arXiv preprint arXiv:2503.21248*, 2025a.

Yujie Liu, Zonglin Yang, Tong Xie, Jinjie Ni, Ben Gao, Yuqiang Li, Shixiang Tang, Wanli Ouyang, Erik Cambria, and Dongzhan Zhou. Researchbench: Benchmarking llms in scientific discovery via inspiration-based task decomposition, 2025b. URL https://arxiv.org/abs/2503.21248.

Zhou Liu, Zhaoyang Han, Guochen Yan, Hao Liang, Bohan Zeng, Xing Chen, Yuanfeng Song, and Wentao Zhang. Datagovbench: Benchmarking llm agents for real-world data governance workflows, 2025c. URL https://arxiv.org/abs/2512.04416.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

Alisia Lupidi, Carlos Gemmell, Nicola Cancedda, Jane Dwivedi-Yu, Jason Weston, Jakob Foerster, Roberta Raileanu, and Maria Lomeli. Source2synth: Synthetic data generation and curation grounded in real data sources, 2025. URL https://arxiv.org/abs/2409.08239.

Maggie, Oren Anava, Vitaly Kuznetsov, and Will Cukierski. Web traffic time series forecasting. https://kaggle.com/competitions/web-traffic-time-series-forecasting, 2017. Kaggle.

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. Discoverybench: Towards data-driven discovery with large language models, 2024. URL https://arxiv.org/abs/2407.01725.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 1–8, 2014a.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation*

(LREC'14), pages 216–223, Reykjavik, Iceland, May 2014b. European Language Resources Association (ELRA). URL https://aclanthology.org/L14-1314/.

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/.

Henry E. Miller, Matthew Greenig, Benjamin Tenmann, and Bo Wang. Bioml-bench: Evaluation of ai agents for end-to-end biomedical ml. *bioRxiv*, 2025. doi: 10.1101/2025.09.01.673319. URL https://www.biorxiv.org/content/10.1101/2025.09.01.673319v2.

Nayantara Mudur, Subhashini Venugopalan, Hao Cui, Paul Raccuglia, Michael Brenner, and Peter Christian Norgaard. FEABench: Evaluating language models on real world physics reasoning ability. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=2z4U9reLm9.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, et al. MLGym: A New Framework and Benchmark for Advancing AI Research Agents. *arXiv preprint arXiv:2502.14499*, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. GPT-4o System Card, 2024a. URL https://cdn.openai.com/gpt-4o-system-card.pdf. Accessed: 2024-06-07.

OpenAI. Introducing gpt-oss, 2024b. URL https://openai.com/index/introducing-gpt-oss/. Accessed: 2024-06-07.

OpenAI. Introducing o3 and o4-mini, 2024c. URL https://openai.com/index/introducing-o3-and-o4-mini/. Accessed: 2024-06-07.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*, 2021.

Yansheng Qiu, Haoquan Zhang, Zhaopan Xu, Ming Li, Diping Song, Zheng Wang, and Kaipeng Zhang. Ai idea bench 2025: Ai research idea generation benchmark, 2025. URL https://arxiv.org/abs/2504.14191.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

Ben Rank, Hardik Bhatnagar, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Measuring ai ability to perform llm post-training, 2025.

Abhinav Rastogi, Adam Yang, Albert Q Jiang, Alexander H Liu, Alexandre Sablayrolles, Amélie Héliou, Amélie Martin, Anmol Agarwal, Andy Ehrenberg, Andy Lo, et al. Devstral: Fine-tuning Language Models for Coding Agent Applications. *arXiv preprint arXiv:2509.25193*, 2025.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10?, 2018. URL https://arxiv.org/abs/1806.00451.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 2023. doi: 10.1038/s41586-023-06924-6.

Kai Ruan, Xuan Wang, Jixiang Hong, Peng Wang, Yang Liu, and Hao Sun. Liveideabench: Evaluating llms' divergent thinking for scientific idea generation with minimal context, 2025. URL https://arxiv.org/abs/2412.17596.

Amrita Saha, Rahul Aralikatte, Mitesh M Khapra, and Karthik Sankaranarayanan. DuoRC: Towards complex language understanding with paraphrased reading comprehension. *arXiv preprint arXiv:1804.07927*, 2018.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

Zachary S Siegel, Sayash Kapoor, Nitya Nagdir, Benedikt Stroebl, and Arvind Narayanan. CORE-Bench: Fostering the Credibility of Published Research Through a Computational Reproducibility Agent Benchmark. *arXiv preprint arXiv:2409.11363*, 2024a.

Zachary S. Siegel, Sayash Kapoor, Nitya Nagdir, Benedikt Stroebl, and Arvind Narayanan. Core-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark, 2024b. URL https://arxiv.org/abs/2409.11363.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's Ability to Replicate AI Research, 2025a. URL https://arxiv.org/abs/2504.01848.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. PaperBench: Evaluating AI's Ability to Replicate AI Research. *arXiv preprint arXiv:2504.01848*, 2025b.

Teague Sterling and John J Irwin. ZINC 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

Zhaojun Sun, Xuzhou Zhu, Xuanhe Zhou, Xin Tong, Shuo Wang, Jie Fu, Guoliang Li, Zhiyuan Liu, and Fan Wu. Surveybench: Can llm(-agents) write academic surveys that align with reader needs?, 2025. URL https://arxiv.org/abs/2510.03120.

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.

Ross Taylor. A Home For Results in ML. Online at https://medium.com/paperswithcode/a-home-for-results-in-ml-e25681c598dc, 2020. Article.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, Andrei Lupu, Roberta Raileanu, Kelvin Niu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H. Miller, Abhishek Charnalia, Derek Dunfield, Carole-Jean Wu, Pontus Stenetorp, Nicola Cancedda, Jakob Nicolaus Foerster, and Yoram Bachrach. AI Research Agents for Machine Learning: Search, Exploration, and Generalization in MLE-bench, 2025. URL https://arxiv.org/abs/2507.02554.

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, 2024. URL https://arxiv.org/abs/2407.18901.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32, 2019.

weco.ai. AIDE: Data science automation technical report. Technical report, weco.ai, 2024. URL https://www.weco.ai/blog/technical-report.

Michael J. Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. SciReplicate-Bench: Benchmarking LLMs in Agent-driven Algorithmic Reproduction from Research Papers. In *Proceedings of the Conference on Language Modeling (COLM)*, 2025a. Published as a conference paper at COLM 2025.

Yanzheng Xiang, Hanqi Yan, Shuyin Ouyang, Lin Gui, and Yulan He. SciReplicate-Bench: Benchmarking LLMs in Agent-driven Algorithmic Reproduction from Research Papers, 2025b. URL https://arxiv.org/abs/2504.00255.

Yijia Xiao, Runhui Wang, Luyang Kong, Davor Golac, and Wei Wang. CSR-Bench: Benchmarking LLM Agents in Deployment of Computer Science Research Repositories, 2025. URL https://arxiv.org/abs/2502.06111.

Yixuan Even Xu, Fei Fang, Jakub Tomczak, Cheng Zhang, Zhenyu Sherry Xue, Ulrich Paquet, and Danielle Belgrave. NeurIPS 2024 Experiment on Improving the Paper-Reviewer Assignment. Online at https://blog.neurips.cc/2024/12/12/neurips-2024-experiment-on-improving-the-paper-reviewer-assignment/#:~:text=This%20year%2C%20for%20NeurIPS%202024,as%20enhance%20reviewer%20diversity%20and, 2022. Article.

Shuo Yan, Ruochen Li, Ziming Luo, Zimu Wang, Daoyang Li, Liqiang Jing, Kaiyu He, Peilin Wu, George Michalopoulos, Yue Zhang, Ziyang Zhang, Mian Zhang, Zhiyu Chen, and Xinya Du. Lmr-bench: Evaluating llm agent's ability on reproducing language modeling research, 2025. URL https://arxiv.org/abs/2506.17335.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc., 2023a.

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*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yunxiang Zhang, Muhammad Khalifa, Shitanshu Bhushan, Grant D Murphy, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. Mlrc-bench: Can language agents solve machine learning research challenges?, 2025. URL https://arxiv.org/abs/2504.09702.

Bingchen Zhao, Despoina Magka, Minqi Jiang, Xian Li, Roberta Raileanu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Kelvin Niu, Shagun Sodhani, Michael Shvartsman, Andrei Lupu, Alisia Lupidi, Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Thomas Foster, Lucia Cipolina-Kun, Abhishek Charnalia, Derek Dunfield, Alexander H. Miller, Oisin Mac Aodha, Jakob Foerster, and Yoram Bachrach. The Automated LLM Speedrunning Benchmark: Reproducing NanoGPT Improvements, 2025. URL https://arxiv.org/abs/2506.22419.

Qiran Zou, Hou Hei Lam, Wenhao Zhao, Yiming Tang, Tingting Chen, Samson Yu, Tianyi Zhang, Chang Liu, Xiangyang Ji, and Dianbo Liu. FML-bench: A Benchmark for Automatic ML Research Agents Highlighting the Importance of Exploration Breadth, 2025. URL https://arxiv.org/abs/2510.10472.

Appendix

A Task Selection

We constructed AIRS-Bench by downsampling a pool $\mathcal{F}$ of approximately 100 tasks to a representative subset $\mathcal{S}$ of 20 tasks. The reduction to 20 tasks was implemented to substantially decrease GPU requirements and enable faster benchmarking. The AIRS-Bench subset was selected to closely mirror the full pool $\mathcal{F}$ according to three key criteria:

- **Agent performance**: each agent's average score on AIRS-Bench is as close as possible to their average score on the full benchmark.
- **Category distribution**: the proportion of tasks from each of the 7 categories in AIRS-Bench is as close as possible to that of the full pool.
- **Relative ranking fidelity**: the ranking of agents by performance is preserved between AIRS-Bench and the full pool.

For each agent a, we compute the mean normalized score $\overline{\mathrm{NS}}^a_{\mathcal{F}}$ over $\mathcal{F}$ and $\overline{\mathrm{NS}}^a_{\mathcal{S}}$ over AIRS-Bench, where

$$\overline{\mathrm{NS}}^a_{\mathcal{F}} = \frac{1}{|\mathcal{F}|}\sum_{t\in\mathcal{F}} \mathrm{NS}^a_t \qquad \overline{\mathrm{NS}}^a = \frac{1}{|\mathcal{S}|}\sum_{t\in\mathcal{S}} \mathrm{NS}^a_t \tag{6}$$

where the normalized score NS^a_t is defined in Equation 2 (see Section 5.2) and is the performance score of agent a on task t. The AIRS-Bench subset is selected to minimize the mean absolute error (MAE) between $\overline{\mathrm{NS}}^a_{\mathcal{F}}$ and $\overline{\mathrm{NS}}^a_{\mathcal{S}}$ across all agents:

$$\mathrm{MAE} = \frac{1}{|A|}\sum_{a\in A}\left|\overline{\mathrm{NS}}^a_{\mathcal{F}} - \overline{\mathrm{NS}}^a_{\mathcal{S}}\right| \tag{7}$$

where A is the set of all agents. To ensure representative coverage, $\mathcal{F}$ is partitioned into four difficulty bands (*easy*, *medium*, *hard*, *expert*), based on their relative ranking by average normalised score (see Section 6.1) and each containing approximately 25 tasks. AIRS-Bench is constructed by sampling a fixed number of tasks from each band. Four candidate difficulty band distributions were evaluated:

- **Uniform**: 5 tasks each from *easy*, *medium*, *hard*, and *expert* bands ($\{5, 5, 5, 5\}$).
- **Medium-skewed**: 4 *easy*, 7 *medium*, 5 *hard*, and 4 *expert* tasks ($\{4, 7, 5, 4\}$).
- **Center-skewed**: 4 *easy*, 6 *medium*, 6 *hard*, and 4 *expert* tasks ($\{4, 6, 6, 4\}$).
- **Medium-heavy**: 3 *easy*, 8 *medium*, 6 *hard*, and 3 *expert* tasks ($\{3, 8, 6, 3\}$).

The final band allocation was selected by choosing the configuration that minimized the mean absolute error (MAE) between agent scores on the subset and the full benchmark. The search for the optimal subset was performed using three subset selection algorithms:

- **Random search**: samples ten thousand candidate subsets, each respecting the band constraints, and retains the subset with the lowest MAE.
- **Simulated annealing**: iteratively swaps tasks within bands, accepting both improvements and, with decreasing probability, worse solutions to escape local minima.
- **Genetic algorithm**: evolves a population of candidate subsets through tournament selection, single-point crossover ($p = 0.7$), and mutation ($p = 0.2$) minimizing MAE over generations.

Across all 12 possible ($\{\text{algorithm}\} \times \{\text{band distribution}\}$) combinations, the best-performing configuration was obtained using the genetic algorithm with a medium-skewed band allocation, achieving a minimum MAE of 4.0×10^{-3}. Other competitive configurations included both uniform and skewed band distributions, with MAE values ranging from 4.6×10^{-3} to 7.9×10^{-3}.

To validate the fidelity of AIRS-Bench, we compared agent mean normalized scores and their 95% confidence intervals on AIRS-Bench versus the full pool $\mathcal{F}$. The results demonstrate that agent rankings and score gaps are faithfully preserved, with overall mean scores and confidence intervals nearly identical between the two sets: the difference in average score between the subset and the full benchmark never exceeds 0.02 in absolute value. This confirms that the selection criterion and stratified sampling approach yield a lightweight benchmark that maintains the discriminative power and ranking structure of the original pool, while allowing efficient evaluation.

B Additional Results

We consider an additional normalized score (see Eq. 2) which employs the identity transform

$$\phi_t(s) = \mathcal{I}(s) = s \tag{8}$$

In contrast to Eq. 3, this approach directly uses raw scores. With this transform, the normalized score NS_t^a linearly reflects the agent's progress between the worst observed solution and the human SOTA for each task. This is simple and interpretable, but may not always reflect meaningful progress when the metric is highly non-linear or when the gap to the optimal score is very small. For completeness, we show the average normalized score using this transform in Figure 12, and provide a breakdown of the scores by difficulty level in Figure 13.

For better readability, Table 6 provides the mapping between task numbers shown on the y-axis of Figures 9-12 and their corresponding task names, as well as their average score across all seeds and agents.

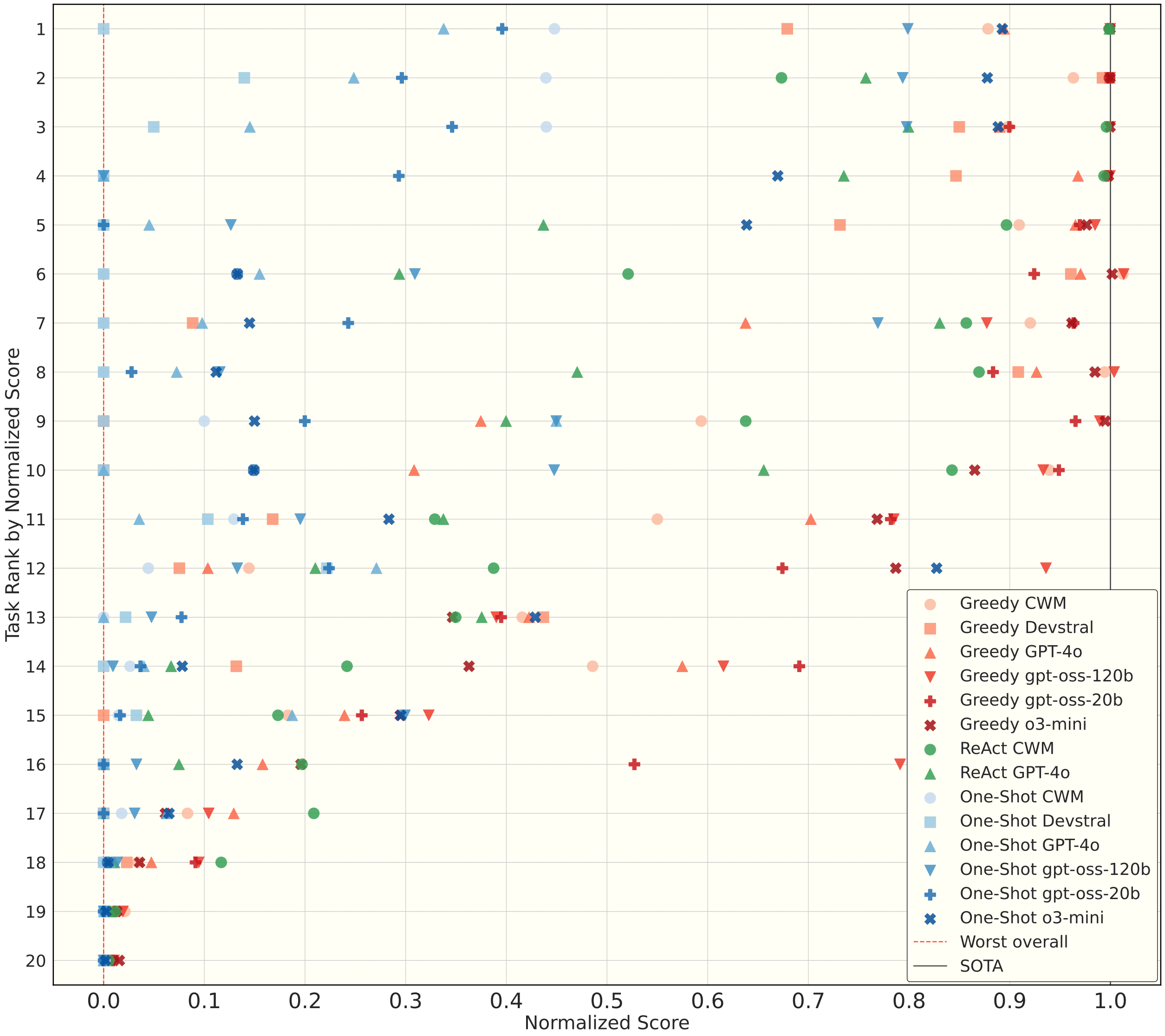


Figure 12 Normalized score per task averaged over seeds, computed according to Equations 2- 8. For each task, the outcome of the worst-performing run is used as the baseline score $s_t^{\min}$. SOTA always corresponds to a normalized score of 1. Tasks are ranked in decreasing order according to the average score across all agents. See Table 6 for the correspondence between tasks ranking and name.

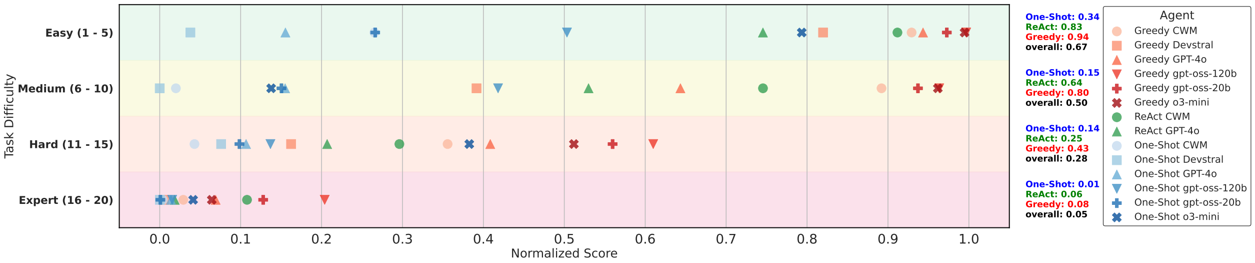


Figure 13 Normalized score per task difficulty level computed according to Equations 2-8. We divide the task ranking of Figure 12 into four categories with decreasing normalized scores: *easy*, *medium*, *hard* and *expert*.

Rank	Task (March of 9's)	NS_t^a	Task (Identity)	NS_t^a
1	TextualClassificationSickAccuracy	0.49	U0MolecularPropertyPredictionQm9MAE	0.74
2	TextualSimilaritySickSpearmanCorrelation	0.49	R2AbsMolecularPropertyPredictionQm9MAE	0.73
3	TimeSeriesForecastingSolarWeeklyMAE	0.39	GMolecularPropertyPredictionQm9MAE	0.71
4	CvMolecularPropertyPredictionQm9MAE	0.38	CvMolecularPropertyPredictionQm9MAE	0.61
5	TimeSeriesForecastingRideshareMAE	0.38	GraphRegressionZincMae	0.55
6	R2AbsMolecularPropertyPredictionQm9MAE	0.35	TextualSimilaritySickSpearmanCorrelation	0.53
7	U0MolecularPropertyPredictionQm9MAE	0.35	TimeSeriesForecastingSolarWeeklyMAE	0.53
8	GMolecularPropertyPredictionQm9MAE	0.34	TextualClassificationSickAccuracy	0.53
9	SentimentAnalysisYelpReviewFullAccuracy	0.31	TimeSeriesForecastingKaggleWebTrafficMASE	0.45
10	ReadingComprehensionSquadExactMatch	0.30	TimeSeriesForecastingRideshareMAE	0.45
11	GraphRegressionZincMae	0.28	SentimentAnalysisYelpReviewFullAccuracy	0.38
12	CoreferenceResolutionWinograndeAccuracy	0.19	ReadingComprehensionSquadExactMatch	0.36
13	QuestionAnsweringEli5Rouge1	0.14	CoreferenceResolutionSuperGLUEWSCAccuracy	0.26
14	QuestionAnsweringDuoRCAccuracy	0.14	CoreferenceResolutionWinograndeAccuracy	0.24
15	CoreferenceResolutionSuperGLUEWSCAccuracy	0.11	QuestionAnsweringDuoRCAccuracy	0.17
16	CodeRetrievalCodeXGlueMRR	0.08	QuestionAnsweringEli5Rouge1	0.15
17	TimeSeriesForecastingKaggleWebTrafficMASE	0.08	CodeRetrievalCodeXGlueMRR	0.05
18	MathQuestionAnsweringSVAMPAccuracy	0.01	MathQuestionAnsweringSVAMPAccuracy	0.03
19	QuestionAnsweringFinqaAccuracy	0.00	QuestionAnsweringFinqaAccuracy	0.01
20	CodeGenerationAPPSPassAt5	0.00	CodeGenerationAPPSPassAt5	0.00

Table 6 Side-by-side ranking of tasks by average normalized score with **march of 9's** and **identity** transforms, as reported in Figure 9, and Figure 12, respectively. Color represents progressively harder tasks, from *easy* to *expert*.

B.1 Task run files

B.1.1 project_description.md

Overview

Task Description

This is a Machine Learning project and your goal is to build a model that solves the project's TASK following the instructions provided below.

TASK: Your task is to solve math word problems. Each example presents a short story followed by a specific question. Your task is to read the text and predict the correct numerical answer. Your predictions will be scored against the Answer column of the test set.

Data

Dataset Structure

The default config of SVAMP has the following structure. Here is a description of the contents of each column including their name, what they contain, and the data type:

```
{
  "ID": string,               # Unique identifier for the problem
  "Body": string,             # The narrative or context text describing the scenario
  "Question": string,         # The actual math question asked based on the Body
  "Equation": string,         # The ground-truth equation used to compute the answer
  "Answer": string,           # The correct numerical solution to the problem (as text)
  "Type": string,             # The problem category/type. Is one of ["Subtraction", "Addition",
                              # "Common-Division", "Multiplication"].
  "question_concat": string   # Concatenation of Body and Question into one text field
}
```

An example entry in the train dataset contains the following:

- ID: "chal-777"
- Body: "There are 87 oranges and 290 bananas in Philip's collection. If the bananas are organized into 2 groups and oranges are organized into 93 groups"
- Question: "How big is each group of bananas?"
- Equation: "(290.0 / 2.0)"
- Answer: "145"
- Type: "Common-Division"
- question_concat: "There are 87 oranges and 290 bananas in Philip's collection. If the bananas are organized into 2 groups and oranges are organized into 93 groups How big is each group of bananas?"

Note: The scoring columns (Equation, Answer, Type) have been removed from the test data. Ensure you preserve the original row order of the test set when producing predictions.
Train data can be loaded using:

```
from datasets import load_from_disk
train_dataset = load_from_disk('./data/train')
```

Test data can be loaded using:

```
from datasets import load_from_disk
test_dataset = load_from_disk('./data/test')
```

Submission File

The submission file should contain the selected answers for the test set. The submission file should be a .csv file named `submission.csv` with the following header:

```
Answer
```

And it should be of shape `(300,1)` — one prediction per row, aligned with the test set order. Each row must contain the numerical answer to the question. Do not include any extra columns or indices.

Evaluation

The evaluation will be performed on the `submission.csv` file you have submitted using the Accuracy metric. Evaluation will be done using the following script:

```python
#!/usr/bin/env python3
import argparse, json
import numpy as np
import pandas as pd
from datasets import load_from_disk


def load_test_answers(path='./data/test_with_labels'):
    """
    Load answers for evaluation (string labels).
    """
    ds = load_from_disk(path)
    if "Answer" not in ds.features:
        raise RuntimeError(
            "Test data does not contain 'Answer'. "
            "Use a labeled copy like './data/test_with_labels'."
        )
    answers = list(ds["Answer"])
    return answers


def evaluate(predictions, labels):
    """
    Returns a dict of metric_name -> value
    """
    correct = np.fromiter(
        (int(p) == int(y) for p, y in zip(predictions, labels)), dtype=bool
    )
    return {"Accuracy": float(correct.mean())}


def _cli():
    p = argparse.ArgumentParser(
        description="Evaluate predictions for SVAMP test split using Accuracy."
    )
    p.add_argument("--submission-file", default="submission.csv",
                   help="Path to CSV file containing predictions.")
    a = p.parse_args()

    print("Loading test set labels...")
    labels = load_test_answers()
    n_test_samples = len(labels)
    print(f"Loaded {n_test_samples} labels.")

    print(f"Loading predictions from: {a.submission_file}")
    try:
        # Assuming CSV has no header and contains only prediction values
        # Adjust if your submission format is different (e.g., has headers, specific columns)
        submission_df = pd.read_csv(a.submission_file, header=0)
        preds = submission_df.values.squeeze()
        if preds.shape[0] != n_test_samples:
            raise ValueError(
                f"Submission file row count ({preds.shape[0]}) "
                f"does not match test set size ({n_test_samples})."
            )
    except FileNotFoundError:
        p.error(f"Submission file not found: {a.submission_file}")
    except Exception as e:
        p.error(f"Error loading submission_file: {e}")

    print("Evaluating predictions...")
    result = evaluate(preds, labels)

    print("\n--- EVALUATION RESULT ---")
    print(json.dumps(result, indent=2))


if __name__ == '__main__':
    _cli()
```

B.1.2 evaluate.py

```python
#!/usr/bin/env python3
import argparse, json
import numpy as np
import pandas as pd
from datasets import load_from_disk


def load_test_answers(path='./data/test_with_labels'):
    """
    Load answers for evaluation (string labels).
    """
    ds = load_from_disk(path)
    if "Answer" not in ds.features:
        raise RuntimeError(
            "Test data does not contain 'Answer'. "
            "Use a labeled copy like './data/test_with_labels'."
        )
    answers = list(ds["Answer"])
    return answers


def evaluate(predictions, labels):
    """
    Returns a dict of metric_name -> value
    """
    correct = np.fromiter(
        (int(p) == int(y) for p, y in zip(predictions, labels)), dtype=bool
    )
    return {"Accuracy": float(correct.mean())}


def _cli():
    p = argparse.ArgumentParser(
        description="Evaluate predictions for SVAMP test split using Accuracy."
    )
    p.add_argument("--submission-file", default="submission.csv",
                   help="Path to CSV file containing predictions.")
    a = p.parse_args()

    print("Loading test set labels...")
    labels = load_test_answers()
    n_test_samples = len(labels)
    print(f"Loaded {n_test_samples} labels.")

    print(f"Loading predictions from: {a.submission_file}")
    try:
        # Assuming CSV has no header and contains only prediction values
        # Adjust if your submission format is different (e.g., has headers, specific columns)
        submission_df = pd.read_csv(a.submission_file, header=0)
        preds = submission_df.values.squeeze()
        if preds.shape[0] != n_test_samples:
            raise ValueError(
                f"Submission file row count ({preds.shape[0]}) "
                f"does not match test set size ({n_test_samples})."
            )
    except FileNotFoundError:
        p.error(f"Submission file not found: {a.submission_file}")
    except Exception as e:
        p.error(f"Error loading submission_file: {e}")

    print("Evaluating predictions...")
    result = evaluate(preds, labels)

    print("\n--- EVALUATION RESULT ---")
    print(json.dumps(result, indent=2))


if __name__ == '__main__':
    _cli()
```

B.1.3 metadata.yaml

```
metric_lower_is_better: false
file_export_globs:
  - submission.csv
container_python_requirements:
  - datasets==4.0.0
evaluate_container_python_requirements:
  - datasets==4.0.0
logging_info:
  name: MathQuestionAnsweringSVAMPAccuracy
  category: Math
  research_problem: Math Question Answering
  output_type: text-generation
  dataset: ChilleD/SVAMP
  metric: Accuracy
  input_columns:
    - question_concat
  scoring_column: Answer
  shape: 300,1
  config: default
  train_split: train
  test_split: test
  custom_gold_labels: false
  custom_rad_class: false
  sota:
    - sota_paper_title: 'Achieving >97% on GSM8K: Deeply Understanding the Problems
        Makes LLMs Better Solvers for Math Word Problems'
      sota_paper_url: https://arxiv.org/pdf/2404.14963v5
      sota_score: 0.942
      sota_notes: DUP is a prompting template. Result provided is for GPT-4 with the
        GUP prompting template.
      sota_year: 2026
      sota_venue: Frontiers of Computer Science
  dataset_paper_url: https://arxiv.org/abs/2103.07191
  estimated_worst_score: 0.0
  optimal_score: 1.0
```

C Harness Setup

	AIRA-dojo	MLGym
Time/number of steps limit	24 hours (up to ∼36 including evaluation time)	1M steps / 24 hours
Can exit early	No	No
evaluate.py file is visible	Yes	Yes
Test set with labels is visible	No	No
Validation script	Agent codes during run	Agent codes during run
Validation splits	Cross-validation (Greedy only)	Classical split 70-30
Scaffold implemented	Greedy (AIDE)	ReAct
Last submission valid always?	No	No
Pretrained models access	Yes	Yes
Num of steps / nodes captured	Yes	Yes
All validation scores captured	Yes	Yes
Every solution scored on test set	Yes	Only final submitted solution
Time limit per solution	4 hours	1 hour
Dummy submission provided	No	No
Num GPUs	1 H200 / run	1 H200 / run
Internet access	Yes (HF_OFFLINE=True, but agent can set it to False)	Yes (can be turned off)
How is evaluate.py provided	In prompt	In shared workspace
Python version	3.10	3.10
Datasets library version	3.5.1 (upgrade to 4.0.0)	4.0.0

Table 7 Resources and constraints comparison between AIRA-dojo and MLGym.

C.1 MLGym 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 1000 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:
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 for evaluation on the test set and allows you to continue
     hill climbing. The test score will be hidden to prevent overfitting, but you’ll get
     confirmation if the submission was valid. Only submit within triple quotes (‘‘‘) should be
     executed - nothing else. Otherwise it will lead to parsing errors
  signature: submit



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 MUST format your output using EXACTLY two fields: DISCUSSION and command.
Your output should ALWAYS include *one* DISCUSSION section and *one* command section in EXACTLY
     this format:

DISCUSSION
[Your reasoning, thoughts, and plan for this step. Be specific about what you’re trying to
     accomplish.]

‘‘‘
[single command to execute]
‘‘‘

CRITICAL FORMATTING RULES:
1. Start with ”DISCUSSION” (no quotes, no extra formatting)
2. Write your thoughts and reasoning
3. Add a single line with three backticks: ‘‘‘
4. Write exactly ONE command
5. End with three backticks: ‘‘‘
6. NO text after the closing backticks
7. NO multiple commands in one response
8. NO multiple code blocks

EXAMPLE OF CORRECT FORMAT:
DISCUSSION
I need to explore the current directory to understand the project structure and identify the data
     files.
```

```
‘‘‘
ls -la
‘‘‘

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.

MACHINE LEARNING WORKFLOW:
Your goal is to achieve the best possible score on a hidden test set. Follow this systematic
     approach:

1. EXPLORATION PHASE:
   - Understand the task by examining data files, baseline scripts, and evaluation metrics
   - Identify the problem type (classification, regression, etc.)
   - Analyze data structure, features, and target variables

2. VALIDATION SETUP:
   - Create a proper train/validation split from the training data
   - If a separate validation set already exists, use it instead of creating your own
   - Set up evaluation code to measure performance on your validation set
   - Your validation code should be separate from evaluate.py provided but it can borrow motivation
     from how you’ll be evaluated with evaluate.py when you do your final submission

3. BASELINE IMPLEMENTATION:
   - Start with the provided baseline script if available
   - Understand the baseline approach and its performance
   - Ensure you can reproduce baseline results

4. ITERATIVE IMPROVEMENT (MOST IMPORTANT PHASE):
   - Make incremental improvements to your model/approach
   - After each change, retrain your model and evaluate on validation set
   - Keep track of what works and what doesn’t
   - Try different approaches: feature engineering, model architectures, hyperparameters
   - CONTINUE EXPERIMENTING even after finding a working solution - this is just the beginning!
   - Use hillclimbing: always try to improve your current best solution
   - Track your best validation score and keep trying to beat it

5. SUBMISSION AND CONTINUED HILL CLIMBING:
   - The ‘submit‘ command now allows you to test your solution without ending the session!
   - When you submit, your code is evaluated on the hidden test set, but the actual score is hidden
     to prevent overfitting
   - You’ll receive confirmation that your submission was valid, then you can continue improving
   - **STRATEGY**: Submit whenever you have a working solution, then keep hill climbing to improve
     it further
   - Use multiple submissions throughout your session to test different approaches
   - **CONTINUE AFTER SUBMIT**: After each submit, keep experimenting with new architectures,
     features, and techniques
   - The goal is to maximize your compute time for hill climbing rather than just finding one
     working solution
```

```
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
     1000 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 be given after every action. Use the remaining steps wisely by
     continuously hill climbing and making multiple submissions to test your improvements!

7. Your each action should take less than 3600 seconds to complete. If your action doesn’t finish
     within the time limit, it will be interrupted.

8. VALIDATION IS KEY: Always use your own validation set (create or use if one exists in data
     directory) to evaluate model performance. Use the ‘submit‘ command to test your submission.csv
     with the hidden test set whenever you have a working solution, then continue improving. Note
     that test scores will be hidden to prevent overfitting, but your submissions will be logged.



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

TASK DESCRIPTION:
<Overview>
<Dataset Structure>
<Evaluation - evaluate.py details>


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.

IMPORTANT SECURITY NOTE ABOUT TEST DATA:
  - You may see references to a ‘test_with_labels/‘ directory in evaluation scripts or documentation
  - This directory is INTENTIONALLY BLOCKED from your workspace for security reasons
  - If you try to access ‘data/test_with_labels/‘, you will find it empty or inaccessible.
  - When you submit your solution, evaluation runs in a SEPARATE secure container that has access
     to the actual test labels
  - DO NOT attempt to access test_with_labels - focus on using ‘data/train/‘ and ‘data/test/‘
     (without labels) for model development
  - Use validation data or cross-validation on the training set to evaluate your model during
     development
  - The ‘submit‘ command will run your code in the secure evaluation environment with actual test
     labels

  OTHER REMARKS
  - Plenty of Huggingface models are pre-downloaded to cache. You can use ‘from transformers.utils
     import hub; print(hub.default_cache_path)‘ to get the path of cache. If you firmly believe that
     the issue is not having the requested model in the cache, then you should be able to also
     download one yourself.
  - Be aware of the running time of the code you write, it should complete within 1 hour.
  - You are provided with a single GPU which you can use for training the models. Take its memory
     into consideration when proposing the solution(s).
  - All the provided input data is stored in ”./data” directory

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!
```

C.2 AIRA-dojo system prompt

The AIRA-DOJO scaffold leverages the following predefined operator set: DRAFT initializes the search process by generating an initial population of candidate solutions. DEBUG attempts to identify and correct errors in buggy solutions. IMPROVE refines valid artifacts to enhance their performance according to the evaluation criteria. While not explicitly defined as an operator, ANALYZE is also used to evaluate the execution output of a generated and executed solution to detect bugs and summarize empirical findings from the results. Detailed below are the prompts for each operator.

C.2.1 Draft

```
# Introduction:
You are a Kaggle grandmaster attending a competition.
In order to win this competition, you need to come up with an excellent and creative plan
for a solution and then implement this solution in Python. We will now provide a description of the
     task.

# Task Description:
{{task_desc}}

{% if memory %}
# Memory:
{{memory}}
{% endif %}

# Instructions:
## Response Format:
```

```
Your response should be a brief outline/sketch of your proposed solution in natural language (3-5 sentences),
followed by a single markdown code block (wrapped in ‘‘‘) which implements this solution and prints out the evaluation metric.
There should be no additional headings or text in your response. Just natural language text followed by a newline and then the markdown code block.

## Solution sketch guideline:
This first solution design should be relatively simple, without ensembling or hyper-parameter optimization.
Take the Memory section into consideration when proposing the design,
don’t propose the same modelling solution but keep the evaluation the same.
The solution sketch should be 3-5 sentences.
Propose an evaluation metric that is reasonable for this task.
Don’t suggest to do EDA.
The data is already prepared and available in the ‘./data‘ directory. There is no need to unzip any files.

## Implementation Guideline:
<TOTAL_TIME_REMAINING: {{time_remaining}}>
<TOTAL_STEPS_REMAINING: {{steps_remaining}}>
The code should **implement the proposed solution**, **print the value of the evaluation metric computed on a hold-out validation set**,
**AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A ‘submission.csv‘ FILE IN THE CURRENT DIRECTORY.**
The code should be a single-file python program that is self-contained and can be executed as-is.
No parts of the code should be skipped, don’t terminate the before finishing the script.
Your response should only contain a single code block.
Be aware of the running time of the code, it should complete within {{execution_timeout}}.
All the provided input data is stored in "./data" directory.
**If there is test data provided for this task, please save the test predictions in a ‘submission.csv‘ file in the "./" directory as described in the task description** This is extremely important since this file is used for grading/evaluation. DO NOT FORGET THE submission.csv file!
You can also use the current directory to store any temporary files that your code needs to create.
REMEMBER THE ./submission.csv FILE!!!!! The correct directory is important too.
The evaluation should be based on 5-fold cross-validation but only if that’s an appropriate evaluation for the task at hand.

## Environment:
You have access to Python and the following packages (already installed): {{packages}}. Feel free to use additional libraries that fit the problem.

# Data Overview:
{{data_overview}}

{% if other_remarks %}
# Other Remarks:
{{other_remarks}}
{% endif %}
```

C.2.2 Debug

```
# Introduction:
You are a Kaggle grandmaster attending a competition.
Your previous solution had a bug and/or did not produce a submission.csv, so based on the
    information below, you should revise it in order to fix this.
Your response should be an implementation outline in natural language, followed by a single
    markdown code block which implements the bugfix/solution.

# Task Description:
{{task_desc}}

{% if memory %}
# Previous debugging attempts:
{{memory}}
{% endif %}

# Previous (buggy) implementation:
{{prev_buggy_code}}

# Execution output:
{{execution_output}}

# Instructions:
## Response Format:
Your response should be a brief outline/sketch of your proposed solution in natural language (3-5
    sentences),
followed by a single markdown code block (wrapped in ‘‘‘) which implements this solution and prints
    out the evaluation metric.
There should be no additional headings or text in your response. Just natural language text
    followed by a newline and then the markdown code block.

## Bugfix improvement sketch guideline:
You should write a brief natural language description (3-5 sentences) of how the issue in the
    previous implementation can be fixed.
Don’t suggest to do EDA.

## Implementation Guideline:
<TOTAL_TIME_REMAINING: {{time_remaining}}>
<TOTAL_STEPS_REMAINING: {{steps_remaining}}>
The code should **implement the proposed solution**, **print the value of the evaluation metric
    computed on a hold-out validation set**,
**AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A ‘submission.csv‘
    FILE IN THE CURRENT DIRECTORY.**
The code should be a single-file python program that is self-contained and can be executed as-is.
No parts of the code should be skipped, don’t terminate the before finishing the script.
Your response should only contain a single code block.
Be aware of the running time of the code, it should complete within {{execution_timeout}}.
All the provided input data is stored in ”./data” directory.
**If there is test data provided for this task, please save the test predictions in a
    ‘submission.csv‘ file in the ”./” directory as described in the task description** This is
    extremely important since this file is used for grading/evaluation. DO NOT FORGET THE
    submission.csv file!
You can also use the current directory to store any temporary files that your code needs to create.
REMEMBER THE ./submission.csv FILE!!!!! The correct directory is important too.
The evaluation should be based on 5-fold cross-validation but only if that’s an appropriate
    evaluation for the task at hand.
```

```
# Data Overview:
{{data_overview}}

# Other remarks
- Huggingface is set to OFFLINE mode by default. If you firmly believe that the issue is not having the requested model in the cache, please set it to ONLINE mode by setting both the environment variables ‘HF_HUB_OFFLINE=0‘ and ‘TRANSFORMERS_OFFLINE=0‘ on top of your code, by importing and using ‘os.environ[...] = ...‘.
- Do not set/force Huggingface to OFFLINE mode as that will NOT fix any issue.
- When a model cannot be found in the ‘timm‘ library, it might be useful to ‘print(timm.list_models())‘.
- If using ‘timm‘ models, remember not to prefix or suffix the model names with datasets such as ‘cifar‘ as this was deprecated.
```

C.2.3 Improve

```
# Introduction:
You are a Kaggle grandmaster attending a competition. You are provided with a previously developed
solution below and should improve it in order to further increase the (test time) performance.
For this you should first outline a brief plan in natural language for how the solution can be improved and
then implement this improvement in Python based on the provided previous solution.

# Task Description:
{{task_desc}}

{% if memory %}
# Memory:
{{memory}}
{% endif %}

# Previous solution:
## Code:
{{prev_code}}

# Instructions:
## Response Format:
Your response should be a brief outline/sketch of your proposed solution in natural language (3-5 sentences),
followed by a single markdown code block (wrapped in ‘‘‘) which implements this solution and prints out the evaluation metric.
There should be no additional headings or text in your response. Just natural language text followed by a newline and then the markdown code block.

## Solution improvement sketch guideline:
The solution sketch should be a brief natural language description of how the previous solution can be improved.
You should be very specific and should only propose a single actionable improvement.
This improvement should be atomic so that we can experimentally evaluate the effect of the proposed change.
Take the Memory section into consideration when proposing the improvement.
The solution sketch should be 3-5 sentences.
Don’t suggest to do EDA.

## Implementation Guideline:
```

```
<TOTAL_TIME_REMAINING: {{time_remaining}}>
<TOTAL_STEPS_REMAINING: {{steps_remaining}}>
The code should **implement the proposed solution**, **print the value of the evaluation metric
    computed on a hold-out validation set**,
**AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A ‘submission.csv‘
    FILE IN THE CURRENT DIRECTORY.**
The code should be a single-file python program that is self-contained and can be executed as-is.
No parts of the code should be skipped, don’t terminate the before finishing the script.
Your response should only contain a single code block.
Be aware of the running time of the code, it should complete within {{execution_timeout}}.
All the provided input data is stored in ”./data” directory.
**If there is test data provided for this task, please save the test predictions in a
    ‘submission.csv‘ file in the ”./” directory as described in the task description** This is
    extremely important since this file is used for grading/evaluation. DO NOT FORGET THE
    submission.csv file!
You can also use the current directory to store any temporary files that your code needs to create.
REMEMBER THE ./submission.csv FILE!!!!! The correct directory is important too.
The evaluation should be based on 5-fold cross-validation but only if that’s an appropriate
    evaluation for the task at hand.

{% if other_remarks %}
# Other Remarks:
{{other_remarks}}
{% endif %}
```

C.2.4 Analyze

```
# Introduction:
You are a Kaggle grandmaster attending a competition.
You have written code to solve this task and now need to evaluate the output of the code execution.
You should determine if there were any bugs as well as report the empirical findings.

# Task Description:
{{task_desc}}

# Implementation:
{{code}}

# Execution output:
{{execution_output}}
```

D Compute Requirements of Benchmarks

Benchmark	GPU / Hardware	Runtime	Budget / Cost	Notes
AIRS-Bench	1×H200 GPUs per run	24h/task	not specified	20 tasks in total, 10–20 runs/task
MLE-Bench	1×A10 GPU	24h/competition	1,800 GPU hours total	75 competitions in total
MLGym-Bench	0–2 GPUs per task (depending on task)	2–4h/task	\$1/run for most LLMs, some are up to \$9	13 tasks in total
RE-Bench	0–6 H100 GPUs (depending on task)	8h/run	\$123/run	7 tasks in total; 3–5 runs/task
ML-Agent-Bench	not specified	0.5–2h/task; max 5h/run	\$60 total	13 tasks in total
SWE-Bench	not specified	not specified	$\leq$ \$0.3 per task; $\sim$\$500 total	2294 tasks in total
CORE-Bench	1×T4 GPU or CPU	2h/task	\$4 per task; max \$6; $\leq$\$500 total	270 tasks in total; 3 trials/task
CSR-Bench	not specified	not specified	not specified	100 GitHub repos in total
Auto-Bench	not specified	not specified	\$365 total	6 tasks in total; 10-20 trials per task
SciReplicate-Bench	1×A100 GPU	not specified	not specified	36 papers broken down to 100 tasks; 3 runs / task
PaperBench	1×A10 GPU	Up to 12h run/paper	\$400 / paper run; \$8k total	20 research papers broken down to 8316 small tasks, 3 runs / paper
ResearchBench	not specified	not specified	not specified	1386 papers each with 3 tasks
Automated LLM Speedrunning	8×H100 GPUs	10h/run; max 20h/run	not specified	19 tasks each with 4 different levels; 3 runs per task+level

Table 8 Summary of compute, runtime, and cost information for recent LLM-agent benchmarks.

E Cached Models

The models' cache available to our agents during the runs consists of the following 193 pretrained models available on HuggingFace, as shown in Table 9. This cache does not contain frontier models, the newest model present is deberta-v3-large released in 2021.

Table 9 HuggingFace models in the run's cache (alphabetically sorted)

Model	Model
ai-forever–ruT5-base	ai4bharat–IndicBERTv2-MLM-only
ai4bharat–indic-bert	albert–albert-base-v2
albert-base-v2	albert-xxlarge-v1
albert-xxlarge-v2	allenai–longformer-base-4096
allenai–scibert_scivocab_uncased	allenai–specter
anferico–bert-for-patents	BAAI–bge-large-en-v1.5
BAAI–bge-small-en-v1.5	bert-base-cased
bert-base-multilingual-cased	bert-base-multilingual-uncased
bert-base-uncased	bert-large-cased
bert-large-uncased	bert-large-uncased-whole-word-masking
bert-large-uncased-whole-word-masking-finetuned-squad	bhadresh-savani–bert-base-uncased-emotion
bhadresh-savani–distilbert-base-uncased-emotion	bhadresh-savani–roberta-base-emotion
camembert–camembert-base	camembert-base
cardiffnlp–twitter-roberta-base	cardiffnlp–twitter-roberta-base-emotion
cardiffnlp–twitter-roberta-base-sentiment	cardiffnlp–twitter-roberta-base-sentiment-latest
cointegrated–rut5-base	cointegrated–rut5-small
cross-encoder–ms-marco-MiniLM-L-6-v2	cross-encoder–nli-deberta-v3-base
cross-encoder–nli-deberta-v3-large	cross-encoder–nli-roberta-base
cross-encoder–stsb-roberta-base	cross-encoder–stsb-roberta-large
deepset–roberta-base-squad2	deepset–roberta-large-squad2
deepset–xlm-roberta-base-squad2	deepset–xlm-roberta-large-squad2
DeepPavlov–rubert-base-cased	distilbert–distilbert-base-cased
distilbert–distilbert-base-uncased	distilbert–distilroberta-base
distilbert-base-cased	distilbert-base-cased-distilled-squad
distilbert-base-multilingual-cased	distilbert-base-uncased
distilbert-base-uncased-distilled-squad	distilbert-base-uncased-finetuned-sst-2-english
distilroberta-base	distilgpt2
dmis-lab–biobert-v1.1	facebook–bart-base
facebook–bart-large	facebook–bart-large-cnn
facebook–bart-large-mnli	facebook–fasttext-en-vectors
facebook–mbart-large-50	facebook–mbart-large-cc25
facebook–wav2vec2-base	facebook–wav2vec2-base-960h
facebook–wav2vec2-large-960h	FacebookAI–roberta-base
FacebookAI–roberta-large	FacebookAI–xlm-roberta-base
FacebookAI–xlm-roberta-large	gpt2
gpt2-large	gpt2-medium
google–bigbird-roberta-base	google–bigbird-roberta-large
google–byt5-base	google–byt5-small
google–electra-base-discriminator	google–electra-large-discriminator
google–electra-large-generator	google–electra-small-discriminator
google–efficientnet-b0	google–efficientnet-b3
google–efficientnet-b4	google–efficientnet-b5
google–efficientnet-b6	google–efficientnet-b7
google–flan-t5-base	google–mobilebert-uncased
google–mt5-base	google–mt5-small
google–muril-base-cased	google–muril-large-cased
google–pegasus-xsum	google–t5-v1_1-small

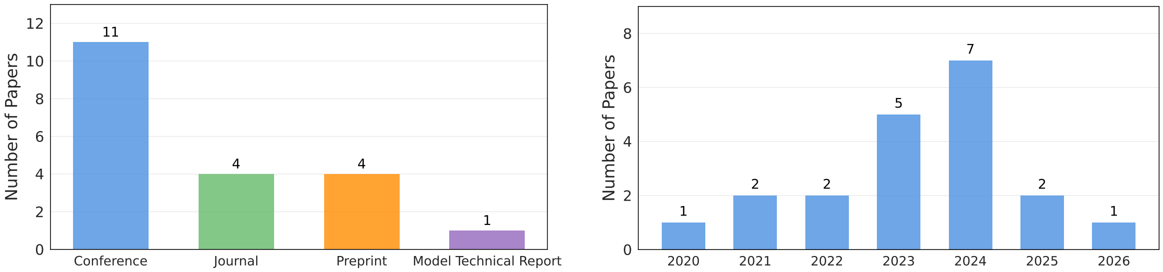
Continued on next page

Table 9 – continued from previous page

Model	**Model**
google–vit-base-patch16-224	google–vit-base-patch16-224-in21k
google–vit-large-patch16-384	google-bert–bert-base-cased
google-bert–bert-base-multilingual-cased	google-bert–bert-base-uncased
google-bert–bert-large-uncased	google-bert–bert-large-uncased-whole-word-masking-finetuned-squad
google–bert_uncased_L-2_H-128_A-2	google–bert_uncased_L-4_H-512_A-8
google-t5–t5-base	google-t5–t5-small
helsinki-nlp–opus-mt-de-en	Helsinki-NLP–opus-mt-en-de
Helsinki-NLP–opus-mt-en-es	Helsinki-NLP–opus-mt-en-fr
Helsinki-NLP–opus-mt-en-ROMANCE	Helsinki-NLP–opus-mt-es-en
Helsinki-NLP–opus-mt-fr-en	Helsinki-NLP–opus-mt-ROMANCE-en
Helsinki-NLP–opus-mt-ru-en	j-hartmann–emotion-english-distilroberta-base
j-hartmann–emotion-english-roberta-large	joeddav–distilbert-base-uncased-go-emotions-student
llm-blender–PairRM	microsoft–codebert-base
microsoft–codebert-base-mlm	microsoft–deberta-base
microsoft–deberta-large	microsoft–deberta-v2-xlarge
microsoft–deberta-v2-xxlarge	microsoft–deberta-v3-base
microsoft–deberta-v3-large	microsoft–deberta-v3-small
microsoft–DialoGPT-medium	microsoft–graphcodebert-base
microsoft–MiniLM-L12-H384-uncased	microsoft–mpnet-base
microsoft–swin-base-patch4-window7-224	microsoft–trocr-base-printed
microsoft–unixcoder-base	microsoft–xtremedistil-l6-h256-uncased
microsoft–xtremedistil-l6-h384-uncased	openai–clip-vit-base-patch16
openai–clip-vit-base-patch32	OpenAssistant–reward-model-deberta-v3-base
OpenAssistant–reward-model-deberta-v3-large	OpenAssistant–reward-model-deberta-v3-large-v2
prajjwal1–bert-mini	prajjwal1–bert-tiny
princeton-nlp–unsup-simcse-roberta-base	ProsusAI–finbert
roberta-base	roberta-base-openai-detector
roberta-large	roberta-large-mnli
s-nlp–roberta_toxicity_classifier	Salesforce–codegen-350M-mono
Salesforce–codet5-base	Salesforce–codet5-base-multi-sum
Salesforce–codet5-large	Salesforce–codet5-small
SamLowe–roberta-base-go_emotions	sberbank-ai–ruT5-base
sberbank-ai–ruT5-large	sentence-transformers–all-distilroberta-v1
sentence-transformers–all-MiniLM-L6-v2	sentence-transformers–all-mpnet-base-v2
sentence-transformers–msmarco-distilbert-base-tas-b	sentence-transformers–paraphrase-albert-small-v2
sentence-transformers–paraphrase-MiniLM-L3-v2	sentence-transformers–paraphrase-MiniLM-L6-v2
sentence-transformers–paraphrase-mpnet-base-v2	sentence-transformers–stsb-mpnet-base-v2
siebert–sentiment-roberta-large-english	stanfordnlp–glove
t5-base	t5-large
t5-small	timm–efficientnet_b4.ra2_in1k
unitary–toxic-bert	unitary–unbiased-toxic-roberta
UrukHan–t5-russian-spell	vectara–hallucination_evaluation_model
vinai–bertweet-base	vinai–bertweet-large
xlnet–xlnet-base-cased	xlnet–xlnet-large-cased
xlnet-base-cased	xlm-roberta-base
xlm-roberta-large	

F Distribution of tasks SOTA venue and year

In Fig. 14 we reprot the breakdown of AIRS-Bench tasks by (a) SOTA publication venue and (b) SOTA publication year. A detailed breakdown of each venue is provided in Table 10.



(a) Breakdown of tasks by SOTA publication venue. **(b)** Breakdown of tasks by SOTA publication year.

Figure 14 Breakdown of tasks by SOTA publication year and venue.

Venue	Count
ICLR	5
Preprint	4
ACL	2
AIMLSystems	1
Frontiers of Computer Science	1
Nature Communications	1
NEURIPS	1
ICML	1
EMNLP	1
Computational Linguistics	1
Model technical report	1
IEEE/ACM Transactions on Audio, Speech and Language Processing	1

Table 10 Breakdown of venues where the SOTA paper was introduced.

G AIRS-Bench Task Description

G.1 CodeGenerationAPPSPassAt5

Solve coding problems by generating five distinct Python programs for each problem. It employs the APPS dataset (Hendrycks et al., 2021), which consists of thousands of real-world coding challenges collected from online platforms, each accompanied by a detailed natural-language problem statement and a starter code template. For each test problem, the problem statement and starter code are provided. Each program is evaluated against a set of hidden test cases, and a prediction is considered correct if at least one of the five submitted programs passes all official test cases. Model performance is assessed using the Pass@5 metric, which measures the fraction of problems solved by at least one of the five attempts.

G.2 CodeRetrievalCodeXGlueMRR

Retrieve relevant code snippets given natural language queries. It uses the CodeXGlue Code Search Adv dataset (Lu et al., 2021), which consists of a large corpus of code functions in Java and a set of queries describing desired functionality in natural language. For each query, the task is to search the corpus and rank code snippets by relevance, aiming to identify the correct code that implements the described functionality. During training and validation, queries are paired with the correct code snippet, while in the test set, only the queries and the code corpus are provided. Model performance is assessed using the Mean Reciprocal Rank (MRR), which measures how highly the correct code is ranked for each query.

G.3 CoreferenceResolutionSuperGLUEWSCAccuracy

Predict whether a pronoun refers or not to something mentioned earlier in the sentence. It uses the SuperGLUE WSC dataset (Wang et al., 2019), which provides sentences with an ambiguous pronoun and a highlighted possible reference. For each example, the agent is given the sentence, the pronoun, and the possible reference. The goal is to predict whether the pronoun refers to that reference (binary classification). Model performance is measured using accuracy, which is the percentage of examples where the correct prediction is made.

G.4 CoreferenceResolutionWinograndeAccuracy

Identify which of two possible options a pronoun in a sentence refers to. It uses the Winogrande dataset (Sakaguchi et al., 2021), which contains sentences with an ambiguous pronoun and two possible answers. For each sentence, select the option that best fills the gap using commonsense reasoning. Model performance is assessed using accuracy, which is the percentage of correct answers.

G.5 CvMolecularPropertyPredictionQm9MeanAbsoluteError

Estimate a molecular property, the heat capacity at constant volume (C_v), from the geometry and atomic composition of a molecule. The agent is required to perform regression based on the 3D coordinates of each atom and its corresponding element. The task utilizes the QM9 dataset (Ramakrishnan et al., 2014), a classic benchmark for molecular property prediction, which spans more than 10 different molecular properties determined using ab-initio density functional theory. Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth C_v values.

G.6 GMolecularPropertyPredictionQm9MeanAbsoluteError

Estimate a molecular property, the Gibbs free energy at $298.15K$ (G), from the geometry and atomic composition of a molecule. The agent is required to perform regression based on the 3D coordinates of each atom and its corresponding element. The task utilizes the QM9 dataset (Ramakrishnan et al., 2014). Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth G values.

G.7 GraphRegressionZincMae

Estimate a molecular property, the constrained solubility of a molecule, from its graph structure. The task utilizes the ZINC dataset (Sterling and Irwin, 2015), a widely used benchmark for graph-based molecular property prediction, which contains thousands of molecular graphs with associated solubility values. The agent is required to perform regression based on the molecular graph, where each molecule is represented as a graph with node features (atom attributes), edge indices (connectivity), and edge attributes (bond types). Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth solubility values.

G.8 MathQuestionAnsweringSVAMPAccuracy

Solve math word problems by reading a short story and answering a specific numerical question. The agent is required to predict the correct numerical answer based on the provided narrative and question, which may involve operations such as addition, subtraction, multiplication, or division. The task utilizes the SVAMP dataset (Patel et al., 2021), a benchmark for evaluating mathematical reasoning and problem-solving abilities in natural language. Each example consists of a description, a question, and the correct answer. Model performance is assessed using accuracy, which is the percentage of examples where the predicted answer matches ground-truth.

G.9 QuestionAnsweringDuoRCAccuracy

Answer questions based on a large context from movie plots. For each example, the agent is provided with the title of a story, a detailed plot summary, and a question about the story. The task is to determine whether the answer to the question is present in the context, and if so, to select the correct answer from a list of candidate answers. The DuoRC dataset (Saha et al., 2018) is used for this task, which contains diverse and challenging reading comprehension questions requiring reasoning over long narrative texts. Model performance is assessed using accuracy, which measures the percentage of questions for which the agent correctly identifies whether an answer exists and, if so, selects the exact answer from the provided candidates.

G.10 QuestionAnsweringEli5Rouge1

Answer open-ended questions using long-form, explanatory responses. For each example, the agent is provided with a question, a detailed context, and is required to generate a comprehensive, human-readable answer. The task utilizes the the ELI5 (Explain Like I'm Five) dataset, containing questions and high-quality, crowd-sourced answers (Fan et al., 2019). Model performance is assessed using the ROUGE-1 F-measure, which evaluates the overlap of unigrams (words) between the generated answer and the reference answer, measuring the quality and relevance of the response.

G.11 QuestionAnsweringFinqaAccuracy

Answer financial reasoning questions based on a combination of textual context and tabular data. For each example, the agent is provided with a question, supporting context, and a table containing relevant financial information. The task utilizes the FinQA dataset (Chen et al., 2021), which is designed to evaluate complex question answering and reasoning over both natural language and structured tables in the financial domain. Model performance is assessed using accuracy, which measures the percentage of questions for which the predicted answer exactly matches the ground-truth, accounting for both numerical and textual equivalence.

G.12 ReadingComprehensionSquadExactMatch

Extract answers to questions from context paragraphs in a reading comprehension setting. For each example, the agent is provided with a title, a context paragraph, and a question about the context. The task is to extract a span of text from the context that answers the question. The dataset uses the SQuAD dataset (Rajpurkar et al., 2016), which is a widely adopted benchmark for machine reading comprehension. Model performance is assessed using the Exact Match metric, which measures the percentage of predictions that exactly match one of the ground-truth answers provided in the dataset.

G.13 R2AbsMolecularPropertyPredictionQm9MeanAbsoluteError

Estimate a molecular property, the electronic spatial extent (R^2), from the geometry and atomic composition of a molecule. The agent is required to perform regression based on the 3D coordinates of each atom and its corresponding element. The task utilizes the QM9 dataset (Ramakrishnan et al., 2014). Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth R^2 values.

G.14 SentimentAnalysisYelpReviewFullAccuracy

Perform sentiment analysis on user-generated reviews from Yelp. For each example, the agent is provided with the text of a Yelp review and is required to predict the corresponding sentiment label, which represents the star rating assigned by the user. The dataset employed derives from the Yelp Dataset Challenge 2015 Asghar (2016), containing reviews labeled as one of five classes: '1 star', '2 stars', '3 stars', '4 stars', or '5 stars' (encoded as 0, 1, 2, 3, or 4). Model performance is assessed using accuracy, which measures the percentage of reviews for which the predicted label exactly matches the ground-truth rating.

G.15 TextualClassificationSickAccuracy

Classify the entailment relationship between two sentences. For each example, the agent is provided with a pair of sentences A and B, and must predict whether the relationship is: *entailment*, i.e. sentence B can be logically inferred from sentence A; *neutral*, there is no clear logical relationship; *contradiction*, sentence B contradicts sentence A. The task uses the SICK dataset Marelli et al. (2014b), a standard benchmark for evaluating models on sentence-level semantic relatedness. Model performance is assessed using accuracy, which measures the percentage of predictions that exactly match the ground-truth label.

G.16 TextualSimilaritySickSpearmanCorrelation

Estimate the semantic relatedness between two sentences by predicting a similarity score from 0 (completely unrelated) to 5 (highly related). For each example, the agent is provided with a pair of sentences, and must output a floating-point score reflecting their degree of semantic similarity. The task uses the SICK dataset Marelli et al. (2014b). Model performance is assessed using the Spearman correlation coefficient between the predicted scores and the ground-truth scores, measuring how well the model's ranking of sentence pairs matches the human-annotated rankings.

G.17 TimeSeriesForecastingKaggleWebTrafficMASE

Perform time series forecasting over the Kaggle Web Traffic dataset, which is part of the Monash Time Series Forecasting Repository. The repository is an extensive collection of time series datasets curated by Monash University and a widely adopted benchmark in the field (Godahewa et al., 2021). The dataset contains 145063 daily time series representing the number of hits or web traffic for a set of Wikipedia pages from 01/07/2015 to 10/09/2017 used by the Kaggle web traffic forecasting competition (Maggie et al., 2017). The goal of the task is to predict the future trajectory of the series by forecasting 59 time steps ahead. Model performance is assessed using the mean absolute scaled error (MASE) between the predicted and ground-truth values in the time series.

G.18 TimeSeriesForecastingRideshareMAE

Perform time series forecasting over the Rideshare dataset, which is part of the Monash Time Series Forecasting Repository (Godahewa et al., 2021). The dataset contains hourly time series representations of attributes related to Uber and Lyft rideshare services for various locations in New York between 26/11/2018 and 18/12/2018. The dataset contains 2304 individual time series, each capturing different aspects of rideshare demand and pricing, including pickup requests, pricing variations, and service availability across different geographic zones and time periods. The goal of the task is to predict the future trajectory of the series by forecasting 48 time steps ahead. Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth values in the time series.

G.19 TimeSeriesForecastingSolarWeeklyMAE

Perform time series forecasting over the Rideshare dataset, which is part of the Monash Time Series Forecasting Repository (Godahewa et al., 2021). The dataset provides weekly aggregated solar power generation and forecast data for a large set of simulated photovoltaic (PV) plants across the United States. The dataset captures the dynamics of solar power generation, including seasonal variations, weather-dependent fluctuations, and geographic diversity across different climate zones. The goal of the task is to predict the future trajectory of the series by forecasting 5 time steps ahead. Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth values in the time series.

G.20 U0MolecularPropertyPredictionQm9MeanAbsoluteError

Estimate a molecular property, the atomization energy at 0 K (U_0), from the geometry and atomic composition of a molecule. The agent is required to perform regression based on the 3D coordinates of each atom and its corresponding element. The task utilizes the QM9 dataset (Ramakrishnan et al., 2014). Model performance is assessed using the mean absolute error (MAE) between the predicted and ground-truth U_0 values.