

MMAU: A Holistic Benchmark of Agent Capabilities Across Diverse Domains

Guoli Yin* Haoping Bai* Shuang Ma*

Feng Nan Yanchao Sun Zhaoyang Xu Shen Ma Jiarui Lu Xiang Kong Aonan Zhang
 Dian Ang Yap Yizhe Zhang Karsten Ahnert Vik Kamath Mathias Berglund Dominic Walsh
 Tobias Gindele Juergen Wiest Zhengfeng Lai George Horrell Xiaoming Wang Jiulong Shan
 Meng Cao† Ruoming Pang† Zirui Wang†

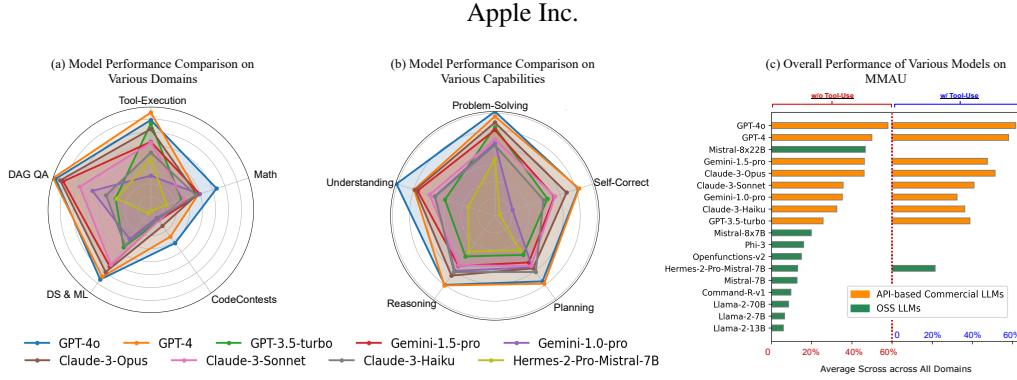


Figure 1: Evaluation results across different models on MMAU. For clarity, this figure includes only a selection of representative models. The domain-centric, capability-centric, and overall evaluation results are aggregated from all 20 tasks in MMAU. For detailed per-task evaluations, please refer to Appendix C.

Abstract

Recent advances in large language models (LLMs) have increased the demand for comprehensive benchmarks to evaluate their capabilities as human-like agents. Existing benchmarks, while useful, often focus on specific application scenarios, emphasizing task completion but failing to dissect the underlying skills that drive these outcomes. This lack of granularity makes it difficult to deeply discern where failures stem from. Additionally, setting up these environments requires considerable effort, and issues of unreliability and reproducibility sometimes arise, especially in interactive tasks. To address these limitations, we introduce the Massive Multitask Agent Understanding (MMAU) benchmark, featuring comprehensive offline tasks that eliminate the need for complex environment setups. It evaluates models across five domains, including **Tool-use**, **Directed Acyclic Graph (DAG) QA**, **Data Science and Machine Learning coding**, **Contest-level programming** and **Mathematics**, and covers five essential capabilities: **Understanding**, **Reasoning**, **Planning**, **Problem-solving**, and **Self-correction**. With a total of 20 meticulously designed tasks encompassing over 3K distinct prompts, MMAU provides a comprehensive framework for evaluating the strengths and limitations of LLM agents. By testing 18 representative models on MMAU, we provide deep and insightful analyses. Ultimately, MMAU not only sheds light on the capabilities and limitations of LLM agents but also enhances the interpretability of their performance. Datasets and evaluation scripts of MMAU are released at <https://github.com/apple/axlearn/blob/main/docs/research/mmau>.

*Equal contribution.

†Senior authors.

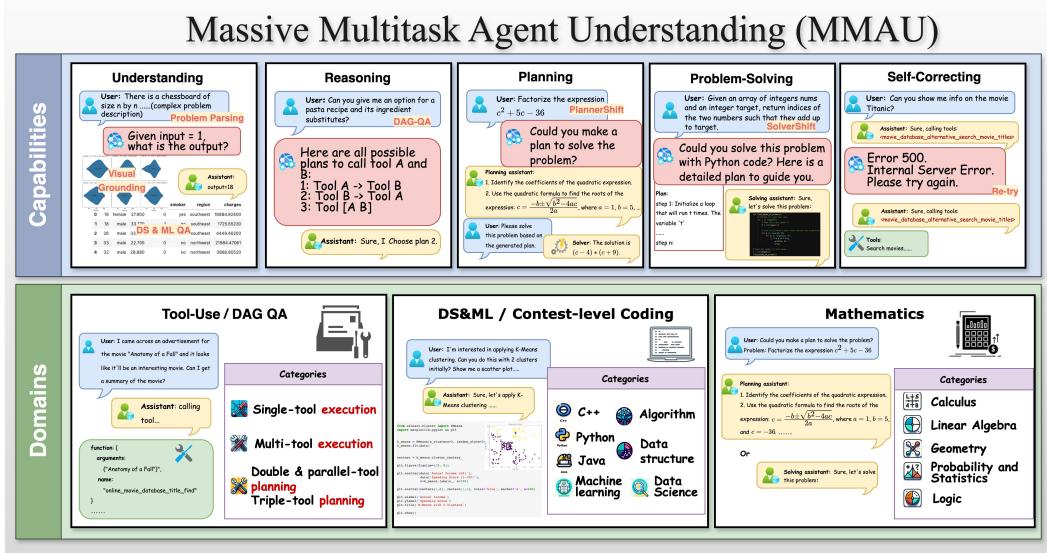


Figure 2: Overview of MMAU. MMAU is designed to provide both capability-centric evaluation (top) and domain-centric evaluation (bottom). It includes over 3K distinct prompts spanning 64 subjects and 5 domains. To evaluate the fundamental capabilities of LLM agents in a disentangled manner, we carefully designed 20 tasks aimed at decomposing these capabilities and assessing performance. Note: For clear visualization, the data examples and prompts here are simplified to illustrate an intuitive example. For the exact data examples and prompts, please refer to the Appendix D B.

1 Introduction

Recent advancements in the field of AI have been marked by significant progresses in the development of LLMs. Particularly, one promising direction along this evolution is the ability of LLMs to perform as human-like agents [1], i.e., understand complex contexts, reason and plan with complicated logic [2–5], make decisions, and utilize tools effectively [6–8]. Consequently, the need for comprehensive benchmarks that evaluate LLMs as intelligent agents has become more and more important.

While existing benchmarks [9–12] evaluate LLM agents by focusing on specific application scenarios and task completion, they struggle to reveal the underlying capabilities driving these outcomes. For example, as shown in Fig. 3, when an LLM encounters a complex math problem, multiple capabilities are required to solve it. By emphasizing task completion, existing benchmarks often obscure whether a failure stems from a lack of comprehension, reasoning, or calculation error. Consequently, these evaluation methods blur the distinctions between different types of failures, hindering our understanding of where the error originates from and limiting our ability to gain deeper insights into the model’s capabilities and make targeted improvements. Additionally, some tasks in existing benchmarks require considerable effort to set up the environments, making a thorough evaluation both expensive and challenging. Furthermore, we observe that tasks, especially interactive ones, are sometimes less stable and reproducible due to the stochasticity of the environment feedback during the evaluation process. This randomness can make it difficult to obtain consistent evaluation results and draw solid conclusions.

To address such limitations, we introduce the Massive Multitask Agent Understanding (MMAU) benchmark. We develop MMAU by identifying five essential capabilities: **Understanding**, **Reasoning**, **Planning**, **Problem-solving** and **Self-correction** across five domains: **Tool-use**, **Directed Acyclic Graph (DAG) QA**, **Data Science & Machine Learning coding**, **Contest-level programming**, and **Mathematics**. As a result, MMAU comprises a total of 3,220 distinct prompts gathered from diverse data sources. These include our in-house human annotations for tool-use, as well as rewritten and curated prompts from open-source datasets such as CodeContest [13], Kaggle [14], and DeepMind-Math [15]. Based on this dataset, we designed 20 tasks across 64 subjects, offering a comprehensive benchmark. To avoid the complexities of environment setup and issues of unreliability, all tasks in MMAU are performed on our 3K static dataset to eliminate potential issues related to environment instability.

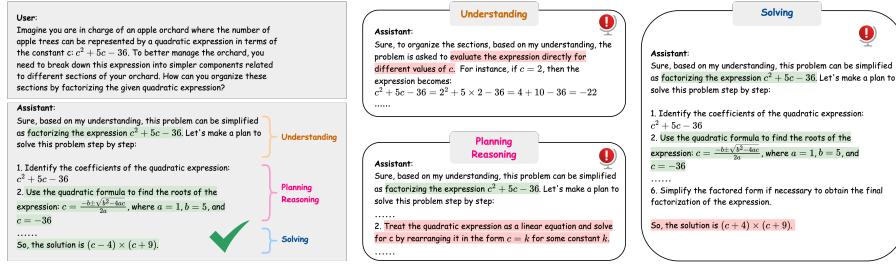


Figure 3: Different error types on a math problem.

We comprehensively evaluate 18 models on MMAU, which include both API-based commercial models and open-source models. In addition to conventional overall comparisons and evaluations tailored to specific application scenarios, we also study the varying capabilities across different models (Figure 1). From our study utilizing MMAU, thorough analysis and insightful analysis arise (Sec. 5). We envision MMAU as a valuable benchmark that not only yields significant observations but also equips the community with deeper insights. Our contributions are summarized as:

- We offer evaluations from both application scenarios and fundamental capabilities, providing a comprehensive framework for understanding the strengths and limitations of LLM agents.
- The evaluation process on MMAU is straightforward and unified on a static dataset, avoiding the instability issues that may arise from interactive evaluations and thus ensuring reliable results.
- We release our evaluation dataset and scripts, aiming to set a new standard for performance assessment in the AI landscape.

We also acknowledge that interactive evaluation is necessary. MMAU does not aim to replace them but rather to complement them by addressing the issues mentioned above. Developing and providing a more stable and easy-to-use benchmark for interactive evaluations is valuable and warrants further studies.

2 Related Work

LLM-based Genelist Agents Researchers have proposed various generalist agent frameworks to create AI assistants that can understand and execute any instruction from users. One pioneering work is Auto-GPT [16], which uses a language model as an AI agent that can break down goals into actionable steps with the aid of auxiliary tools. Integrating language models into multi-agent collaboration systems [17] is a cutting-edge research area. Frameworks like AutoGen [18], LangChain [19], Camel [20], AGENTS [21], AutoAgents [22], and XAgent [23] have explored different approaches to enable communicative agents to collaborate autonomously, facilitate practical applications, ensure control and customization, dynamically generate specialized agents, and manage complex tasks effectively.

Agent Benchmarks The need for rigorous benchmarks also arises in response to the surge in LLM-based agents. Varying benchmarks are created to gauge agent capability on tasks inspired by real-world use cases. The Berkeley Function Calling Leaderboard [24], NexusRaven V2 Function Calling Benchmark [25], ToolBench [26], StableToolBench [27], and API-BLEND [28] seek to evaluate the capability of LLM agent to plan and perform function calls. Webshop [29], WebArena [30], Mind2Web [31], MiniWoB++ [32], and VisualWebArena [33] focus on the agent’s ability to browse and interact with a web environment. A line of benchmarks consider the universal presence of user interface (UI) and envision UI automation agents, including PixelHelp [34], MetaGUI [35], MoTIF [36], AITW [37], and OmniACT [38]. SWE-bench [39] tests agent capability to solve real-world software engineering problems.

While each benchmark tends to focus on a specific application, a generalist agent should be able to perform well on a wide range of tasks. AgentBench [12] consolidates tasks covering coding, game, and math into a single systematic benchmark. AgentBoard [11] evaluates agent capabilities under web browsing, tool use, embodied AI, and game domains. However, both benchmarks require containerized environments to run and need involved effort to implement new tasks. As a result, simply stacking tasks would lead to diminishing returns in comprehensiveness. MMAU takes a step

Table 1: Comparison of benchmarks in evaluating core capabilities of LLM agents. ‘‘En.’’ and ‘‘Dis.’’ represent entangled and disentangled, specifically. Understand.: understanding, Reason.: reasoning, Plan.: planning, Prob.-solv.: problem-solving, Self-corr.: self-correction, MM: multimodal grounding.

Benchmarks	Understand.		Reason.		Plan.		Prob.-solv.		Self-corr.	MM
	En.	Dis.	En.	Dis.	En.	Dis.	En.	Dis.		
AgentBench [12]	✓	✗	✓	✗	✓	✗	✓	✗	✓	✓
AgentBoard [11]	✓	✗	✓	✗	✗	✓	✓	✓	✗	✗
PlanBench [40]	✓	✗	✓	✓	✓	✓	✗	✗	✗	✗
MMLU [9]	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗
MMMU [10]	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
MMAU	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

back and considers a range of core agent capabilities. Based on the core capabilities, MMAU provides a range of tasks that are designed to produce decoupled metrics over the core capabilities.

Table 1 compares the supporting capabilities of various benchmarks. PlanBench [40], specifically designed for benchmarking planning capabilities in LLM agents, also supports disentangled evaluation of reasoning and planning. AgendaBoard [11] provides a manually labeled subset for evaluating disentangled capabilities, focusing on interactive agents, such as spatial navigation and world modeling. In contrast, MMAU evaluates more fundamental and essential capabilities for LLM agents. Overall, MMAU offers a more comprehensive evaluation of fundamental capabilities in a disentangled manner.

3 The MMAU Benchmark

To introduce MMAU, we will start with an overview of all included capabilities 3.1. We will then provide detailed explanations of how each task was designed and how the dataset across different domains was constructed 3.2.

3.1 Capabilities in MMAU

Below, we introduce the capability definitions and the key tasks used to evaluate them. A complete task-capability mapping can be found in the Appendix 6.

Understanding is a fundamental capability required of an intelligent agent. In MMAU, we evaluate an agent’s understanding in different aspects, including: *complex instruction following, user intent understanding, statistics parsing, and visual grounding*.

Reasoning and Planning reflect an agent’s thought process and ability to infer logically from complex factors. Although reasoning and planning has been recognized in many works, it is often described as a general ability, compounded with other skills, which limits deep investigation. In MMAU, we address this challenge with the task `planner-shift`, designed to decompose reasoning and planning capabilities from other factors. Unlike standard end-to-end evaluations, `planner-shift` divides the solution generation into two stages. In stage 1, a `planner model` generates a high-level plan, providing a strategy to solve the given problem without hinting at the final solution. In stage 2, a `solver model` is given the original problem along with the plan to solve it. This approach isolates the planning and reasoning processes from problem-solving. To test planning and reasoning capabilities, we vary only the `planner model` while using the same `solver model`, ensuring that performance differences reflect the planning and reasoning capabilities. The task design diagram is shown in Figure 4.

Problem-solving focuses on measuring an agent’s ability to successfully implement or execute a task, assuming it has already understood and planned the strategy well. To address this, we design a task called `solver-shift`, similar to `planner-shift`, which also performs a two-stage generation. However, `solver-shift` keeps the `planner model` constant and varies only the `solver models` to reflect differences in problem-solving skills, as shown in Figure 4. In MMAU we use the tasks of `planner-shift` and `solver-shift` in domains of [Contest-level coding](#) 3.2.3 and [Math](#) 3.2.4.

Self-correction is another core capability for an intelligent agent. It reflects the agent’s ability to identify errors, learn from its environment and past behaviors, and correct itself to eventually

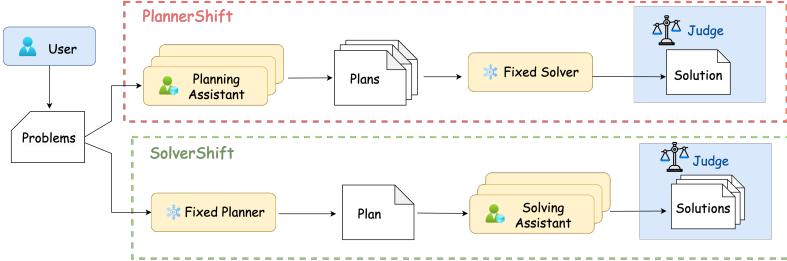


Figure 4: Construction of planner-shift task and solver-shift task.

overcome obstacles and achieve its task. In MMAU, we evaluate this capability by specifically designing self-correction tasks across different domains.

3.2 Dataset Construction

The construction of MMAU encompasses both breadth and depth of data, as illustrated in Table 4. Our dataset is constructed from heterogeneous sources: 1) our in-house tool-use data, used for tasks under **tool-use** and **DAG-QA**; 2) Kaggle [14] datasets, which we rewrite to design tasks for **DS & ML** coding; 3) **CodeContest** [13], used for tasks under contest-level coding; and 4) DeepMind-math [15], used for **math** tasks. MMAU involves curating and rewriting these data sources. In the following section, we will explain how we leveraged these source data to construct MMAU.

3.2.1 Tool-Use

We curated an in-house dataset for tool-use with conversation trajectories following the standard tool-use (a.k.a. function-calling) protocol. We select from a subset of RapidAPI Hub³ functions and ask human annotators to create realistic scenarios with user queries, ground truth function calls and actual function returns from the RapidAPI endpoints. In total, our in-house tool-use dataset consists of 409 single-step (**task:single-tool-use**) and 258 multi-step tool-use conversations (**task: multi-turn multi-tool-use**). Out of the 409 single-step tool use conversations, 225 require making parallel tool calls (**task: parallel-tool-use**). Figs. 10, 11 and 12 show some of these examples. We adapt this dataset for the following tasks.

Task: Tool-use Benchmarking agent tool-use following the standard protocol requires an interactive environment. To simplify the evaluation process, we instead evaluate the model’s response at each assistant turn (i.e., where a function call is expected), conditioning on the ground-truth versions of all previous user or assistant turns. For evaluation, we check if the model’s tool call matches that of the ground truth, i.e. calling the same function and the same parameters.

Task: DAG QA In this task, a user presents a set of requirements to which the LLM must respond by selecting and ordering a sequence of tool invocations from multiple choices provided. This design examines whether the model can identify the relevant tools and deduce the correct dependencies between them. The prompt enumerates the possible tool use orderings from which the LLM agent is asked to pick one, and the label is derived from the ground truth function call sequence. For example, we transform the multi-step example in Figure 11 into a DAG QA task as shown in Figure 14

Task: Tool-use Self-correction From the tool-use dataset described above, we derive two classes of errors to test the model’s self-correction ability:

Temporary error simulates a tool that is temporarily unavailable. From the ground truth messages [user queries, tool-calls and tool responses], we substitute a random **temporary error** (e.g. “429: *too many requests*”, “504: *gateway timeout*”) in place of the tool response.

Incorrect call simulates a previous tool call or response containing an error. We mutate the ground truth tool-call to be incorrect by changing the arguments or the function name, issue the modified call, and save the updated tool response. Given the message history of [user queries, mutated calls, updated tool responses], the model is expected to retry with the correct call.

The evaluation metric is the exact match accuracy of the function name and arguments against the ground truth.

³<https://rapidapi.com/hub>

3.2.2 Data Science and Machine Learning

We leverage the Meta Kaggle Code dataset [14] and curate 28 Python notebook-style conversations, with 123 conversation turns. Each turn begins with a user request for code generation. Among all requests, 83 requests expect text-based outputs from code and 40 requests expect image outputs. Due to the open-ended nature of code generation, we created multiple-choice questions that require information from successful code execution to fully address, resulting in 207 text-based questions and 121 image-based questions. Figure 5 shows an example turn with multiple choice questions. We report QA accuracy as the main metric and vary the combination of code model and QA model to produce different evaluation settings.

Task: E2E Code Generation and QA In this setting, we aim to gauge the overall capability. the evaluated model is responsible for both code generation and QA.

Task: Code Generation and GPT-4 QA In this setting, we isolate the code generation capability of the model. After generating code from the evaluated model, we adopt a strong multimodal model (GPT-4 [41]) to serve as control and perform QA based on code execution outputs.

Task: QA from Oracle Code In this setting, we specifically focus on the textual and visual understanding proficiency of the model decoupled from code generation. We obtain oracle output by executing ground truth code implementation and then pass to the evaluated model to perform QA.

Task: DS & ML Self-Correction This setting is similar to the E2E setting, however, whenever code execution fails, we use the execution error message to prompt an additional code generation turn.

3.2.3 Contest-Level Coding

For contest-level coding problems, we select 261 problems from the Valid and Test splits of the CodeContests dataset [13] which includes competitive programming problems. We adapt these 261 problems for the following tasks.

Task: E2E Standard In this task, models are challenged with a variety of coding problems. The effectiveness of the solutions is measured by executing the code against all predefined test cases [13]. All CodeContests results reported in this paper are based on pass@K ($K=5$) accuracy.

Task: Planner-shift and Solver-shift As introduced in Section 3.1, we use thses two tasks to extensively measure the agent’s capability in planning and problem-solving, respectively. We evaluate both of these tasks by generating K Python code solutions and verifying their pass rate.

Task: Problem Parsing Unlike the other tasks, this task does not require the model to write or execute any code. Instead, given a problem statement and associated test cases, the model is only tasked with predicting the outputs for these test cases. An example is shown in Figure 15. Our rationale is that if a model truly grasps the problem, including its complex instructions and user intent, it should be able to accurately predict the outputs based on its understanding alone. We use match accuracy as the evaluate metric.

Task: CodeContest Self-correction For the E2E standard task above, we collect the error messages of each candidate solution if it does not pass some test cases, including 4 types of errors: *empty*

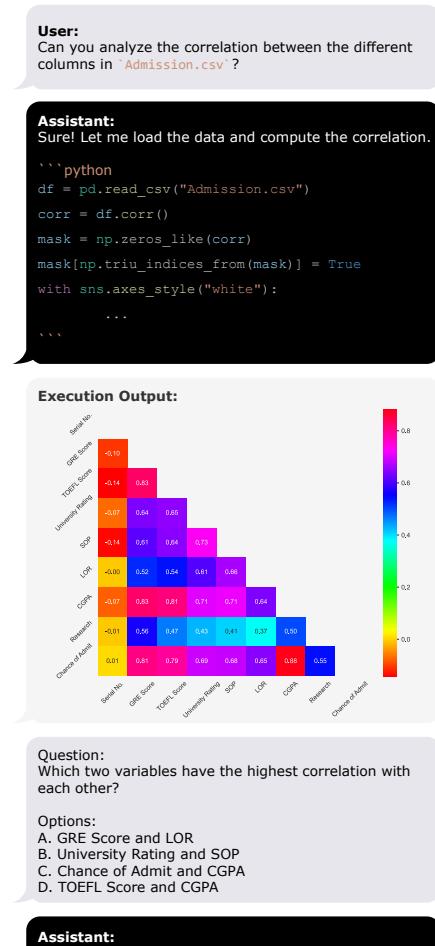


Figure 5: A Multi-turn coding and QA example for Data science and Machine learning.

Table 2: Domain-centric evaluation results. All values are reported as percentages (%). Models that do not support tool use are labeled as N/A, while models not supporting multi-tool tool-use tasks are marked by *⁴. The bolded and underlined numbers indicate the 1st and 2nd highest performances in each category.

Model	Tool-use	DAG	DS&ML	CodeContests	Math
GPT-4o [42]	69.33	77.38	66.52	31.80	53.40
GPT-4-Turbo [43]	75.13	79.38	<u>63.90</u>	<u>25.67</u>	38.57
GPT-3.5-Turbo	66.63	28.60	35.79	10.34	25.00
Gemini-1.5-pro [44]	52.43	71.18	54.02	10.34	<u>39.70</u>
Gemini-1.0-pro [45]	26.32	47.45	27.87	7.66	39.40
Claude3 Opus [46]	62.39	73.61	59.45	15.33	37.40
Claude3 Sonnet [46]	51.92	57.65	52.44	10.34	26.80
Claude3 Haiku [46]	44.14	36.36	33.84	8.81	36.60
Mixtral-8x22B-v0.1 [47]	N/A	72.51	31.10	9.20	50.00
Mixtral-8x7B-v0.1 [47]	N/A	30.82	<u>13.48</u>	1.92	<u>21.70</u>
Mistral-7B-v0.2 [48]	N/A	<u>34.15</u>	2.80	0.38	9.55
Phi-3-mini4K-instruct [49]	N/A	23.95	0.84	2.30	21.00
Openfunctions-v2 [50]	26.53*	25.28	3.84	<u>8.05</u>	15.75
Hermes-2-Pro-Mistral-7B [51]	39.48	27.94	3.29	0.77	12.78
Command R [52]	28.29*	22.28	0.00	4.21	8.21
LLama2-70B [53]	N/A	19.73	0.00	0.00	8.43
LLama2-13B [53]	N/A	17.96	0.00	0.00	4.10
LLama2-7B [53]	N/A	20.84	0.00	0.00	3.92

solution, compilation error, runtime error, and wrong outputs. Then, we follow the setup as what we used in **Tool-use self-correction** (Sec. 3.2.1) to append the error content as a feedback user message to the message list, and ask the model to try again. We still measure the pass@K metric by generating K candidates independently.

3.2.4 Mathematics

The source data in the domain of math, derived from DeepMind-Math [15], consists of 1,000 carefully curated math problems spanning 56 subjects, including calculus, geometry, statistics, and etc.

Task: E2E Standard We adhere to standard protocol by incorporating a Chain-of-Thought (CoT) [3] into the prompt to generate the answers end-to-end, and using accuracy as the evaluation metric.

Task: Planner-shift and Solver-shift As introduced in Sec. 3.1, we use the two tasks to assess the model’s planning and problem-solving abilities in a two-stage manner, avoiding confounding influences from other capabilities. The prompts used for each task can be found in the Appendix B.

Task: Comprehend+. To better isolate and assess the understanding capability without excessive interference from other skills, we have devised a new task named Comprehend+. Our hypothesis is that problems that are straightforward mathematically but complex in their descriptions rely more heavily on understanding capabilities. To test this, we first selected a subset containing only the mathematically simpler problems, and then use an LLM to create new math problems that feature more complex descriptions or harder problem statements but retain the same underlying mathematical constructs from each data sample. A rewritten math problem example is shown in Fig. 13. After curation and verification, we finalize 676 newly created problems for Comprehend+. Please refer to Appendix A.5 for details of dataset creation.

4 Evaluation

We comprehensively evaluate 18 models on MMAU. All evaluation model details are listed in Table 5. For easier reading, the main paper presents only the aggregated evaluation results. For the evaluation results over all 20 tasks, please refer to Appendix C.

4.1 Domain-centric Evaluation and Analysis

As shown in Table 2, there is a clear performance gap between API-based commercial models and open-source models across all evaluation domains. Among the commercial models, the GPT-4 family

Table 3: Capability-centric evaluation results. All values are reported as percentages (%). Models that do not support tool use are labeled as N/A, while models not supporting multi-tool tool-use tasks are marked by *⁵. The bolded and underlined numbers indicate the 1st and 2nd highest performances in each category.

Model	Problem-Solving		Understanding	Reasoning	Planning	Self-correct	
	w/o Tool-Use	w/ Tool-Use				w/o Tool-Use	w/ Tool-Use
GPT-4o [42]	56.12	61.11	60.63	50.47	47.90	43.65	51.56
GPT-4-Turbo [43]	48.07	<u>58.29</u>	<u>49.78</u>	50.88	49.59	<u>40.86</u>	51.86
GPT-3.5-Turbo	42.84	51.83	30.78	29.38	28.27	21.23	32.38
Gemini-1.5-pro [44]	49.04	50.32	47.63	36.28	33.77	34.69	36.66
Gemini-1.0-pro [45]	<u>52.84</u>	42.82	37.05	39.88	37.31	15.96	10.91
Claude3 Opus [46]	49.98	54.67	49.03	44.10	38.84	38.47	44.16
Claude3 Sonnet [46]	40.47	44.79	40.22	36.57	36.92	31.01	37.05
Claude3 Haiku [46]	39.77	41.42	36.85	42.09	42.09	20.44	30.01
Mixtral-8x22B-v0.1 [47]	49.04	N/A	44.39	44.92	38.02	18.00	N/A
Mixtral-8x7B-v0.1 [47]	<u>33.50</u>	N/A	<u>27.98</u>	<u>27.57</u>	<u>30.67</u>	<u>8.26</u>	N/A
Mistral-7B-v0.2 [48]	21.87	N/A	9.01	21.45	21.22	1.93	N/A
Phi-3-mini4K-instruct [49]	22.94	N/A	14.92	28.33	27.45	2.04	N/A
Openfunctions-v2 [50]	20.89	29.43*	11.20	23.30	24.76	2.61	22.77*
Hermes-2-Pro-Mistral-7B [51]	29.26	33.12	16.54	25.95	24.96	2.04	2.99
Command R [52]	22.17	31.30*	19.47	22.72	22.95	0.35	18.84*
Llama2-70B [53]	24.23	N/A	6.32	15.30	14.55	0.34	N/A
Llama2-13B [53]	12.86	N/A	2.69	13.39	13.15	0	N/A
Llama2-7B [53]	15.92	N/A	2.38	14.25	13.26	0	N/A

(including GPT-4o and GPT-4) consistently outperforms other models. In math and contest-level coding, GPT-4o demonstrates a significant advantage. Additionally, Claude3-Opus and Gemini-1.5-pro perform reasonably well. While for open-source models, a significant number do not support tool-use. Among those that do, Hermes-2-Pro-Mistral-7B demonstrates strong tool-use performance. For models that do not support tool use, Mixtral-8x22B-Instruct-v0.1 performs surprisingly well in math and DAG-QA, demonstrating its strong reasoning and planning capability. Additionally, Phi-3 performs well in math considering its model size. The Llama2 family, however, struggles with challenging coding tasks.

4.2 Capability-centric Evaluation and Analysis

As introduced in Sec.3.1, we designed tasks to decompose core capabilities from standard evaluations, allowing MMAU to offer a unique dimension of evaluation. Each capability includes tasks spanning different domains. To provide overall capability-centric evaluation results for each model, we aggregate tasks under each capability using a weighted average. Detailed task-capability mappings and the calculation method can be found in Appendix A. The overall capability-centric evaluation results are shown in Table 3.

Notably, for the capability of **Understanding**, GPT-4o significantly outperforms other models, demonstrating its superior capability in handling long contexts, complex user instructions, and capturing (sometimes implicit) user intents. Additionally, GPT-4, Gemini-1.5-pro, and Claude3-Opus also exhibit reasonably strong understanding capabilities. For the capabilities of **Reasoning** and **Planning**, the GPT-4 family shows the strongest performance. When examining the capability of **Problem-solving**, the performance gap is not significantly large. This trend suggests that when provided with "oracle" plans, solving a task may be less challenging. While models' problem-solving capabilities vary, most can perform these tasks reasonably well, indicating that this capability may be more universally achievable among different models. On the contrary, for **Self-correction**, we observe a significant gap among models. Among open-source models, aside from Mixtral-8x22B, others do not seem to possess the skill to reflect on and correct their own errors effectively. These evaluation results highlight that self-correction is a critical capability needing further research and development to advance the field.

5 Analysis and discussion

How does planning impact the performance? One interesting finding emerges from the results of our designed tasks, Planer-Shift and Solver-Shift on **Math**. As shown in Table 7, we find

⁴The current reported and documented prompt templates of these models do not support multi-tool execution. However, it can still be possible that models can call multiple tools with proper adaptation.

that high-quality planning can boost performance for all models on Math. For example, Command R’s performance increases from 8.21% to 33.33%, and Llama-2-70B’s from 8.43% to 32.10%. Even already strong models, such as Mixtral-8x22B, saw improvement from 50% to 60.02%. Interestingly, using the model itself as the planner also improves performance, e.g. bumping GPT-4o from 53.4% to 61.2%. This shows that explicitly instructing the model to first develop a high-level strategy and then solve the problem based on that strategy can be a promising approach to further enhance performance.

Do different capabilities present varying levels of difficulty for models to achieve? Our evaluation results reveal that different capabilities indeed present varying levels of difficulty for models to achieve. As what we mentioned in Sec. 4.2, **problem-solving** capabilities exhibit a smaller performance gap among models, suggesting that problem-solving is a more universally achievable capability across different models. However, **self-correction** capabilities present a significant challenge, with a notable performance gap observed among models, and many open-source models lacking effective self-correction skills. These findings suggest that while some capabilities like **problem-solving** are more readily attained by current models, others, such as **self-correction** and **planning** pose greater challenges and are vital areas for future advancements in the field.

Do balanced capabilities indicate the path to a generalist agent? From Figure 1, we observe that strong models, such as the GPT-4 family, exhibit balanced performance across all capabilities, demonstrating their robust and versatile nature. This balance indicates that improvements in one area likely enhance performance in others, highlighting a high correlation and interdependence among these capabilities. Conversely, models that perform poorly in one capability tend to struggle across the board, suggesting underlying weaknesses in their architecture or training strategy.

Are larger models always better? Another interesting finding arises when comparing the MistralAI families and Llama-2 families. For the MistralAI models, we consistently observe performance gains with increasing model size across domains. However, this trend does not apply to the Llama-2 families. In code-related domains (DS & ML, CodeContest), all size variants of Llama-2 perform poorly. Surprisingly, in the DAG-QA domain, the Llama-2-7B model performs better than its larger counterparts. This observation is consistent with findings from AgentBench [12], further validating that training strategies and model architectures also influence the scaling law.

6 Conclusion

In this paper, we introduce the Massive Multitask Agent Understanding (MMAU) benchmark. By evaluating models based on both application scenarios and fundamental capabilities, MMAU provides a comprehensive and in-depth test bed for reliable and thorough studies. By designing 20 tasks to decompose capabilities beyond standard evaluation benchmarks, MMAU offers more granular insights into the strengths and limitations of these models.

Limitations and future work. The current scope of MMAU, while broad, does however not encompass all possible domains relevant to LLM agents, such as interactive environments which are also critical yet challenging. Future iterations of MMAU should aim to include interactive tasks to provide a more holistic evaluation. This expansion will require the development of reliable, stable, and user-friendly interactive environments. Moreover, as we expand to include more domains, it will be essential to incorporate additional capabilities such as retrieving, memorizing, sequential decision-making, etc. Our current approach to capability decomposition, though insightful, still faces challenges in disentangling compound capabilities. Future research should focus on developing more effective methods for decomposing and evaluating these capabilities to further refine the benchmark.

Ethics and Societal Impacts. Research on LLM agents must consider potential ethical concerns and negative societal impacts. The MMAU benchmark aims to provide a thorough and transparent evaluation framework, but it is crucial to ensure that these evaluations do not inadvertently reinforce biases or propagate harmful content. We are also careful in detecting and mitigating any personally identifiable information or offensive content within our datasets and prompts.

Acknowledgement

We acknowledge and thank Mark Lee who helped with our code review and release.

References

- [1] Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.
- [2] 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. *Advances in Neural Information Processing Systems*, 36, 2024.
- [3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [4] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.
- [5] Hao Liu, Carmelo Sferrazza, and Pieter Abbeel. Chain of hindsight aligns language models with feedback. *arXiv preprint arXiv:2302.02676*, 2023.
- [6] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.
- [7] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.
- [8] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueling Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024.
- [9] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.
- [10] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. *arXiv preprint arXiv:2311.16502*, 2023.
- [11] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*, 2024.
- [12] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.
- [13] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittweiser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustín Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *arXiv preprint arXiv:2203.07814*, 2022.
- [14] Jim Plotts and Megan Risdal. Meta kaggle code, 2023.
- [15] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.
- [16] Significant Gravitas. Auto-gpt: An autonomous gpt-4 experiment, 2023. URL <https://github.com/Significant-Gravitas/Auto-GPT>, 2023.
- [17] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2023.
- [18] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.
- [19] Harrison Chase. LangChain, October 2022.

- [20] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*, 2023.
- [21] Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruiyu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Xiangru Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. Agents: An open-source framework for autonomous language agents, 2023.
- [22] Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation, 2024.
- [23] XAgent Team. Xagent: An autonomous agent for complex task solving, 2023.
- [24] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.
- [25] Nexusflow.ai team. Nexusraven-v2: Surpassing gpt-4 for zero-shot function calling, 2023.
- [26] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.
- [27] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2024.
- [28] Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis A. Lastras. Api-blend: A comprehensive corpora for training and benchmarking api llms. *ArXiv*, abs/2402.15491, 2024.
- [29] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *ArXiv*, preprint.
- [30] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.
- [31] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023.
- [32] Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, pages 9466–9482. PMLR, 2022.
- [33] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.
- [34] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. Mapping natural language instructions to mobile ui action sequences. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*, 2020.
- [35] Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: Towards multi-modal conversational agents on mobile gui. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6699–6712, 2022.
- [36] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments.
- [37] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control, 2023.
- [38] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web, 2024.

- [39] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.
- [40] Karthik Valmeeakam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.
- [41] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [42] OpenAI. Gpt-4o. <https://openai.com/index/hello-gpt-4o/>, 2024.
- [43] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffee, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Lukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emry Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.
- [44] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittweiser, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.
- [45] Gemini Google, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *ArXiv preprint abs/2312.11805*, 2023.
- [46] AI Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 2024.

- [47] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.
- [48] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *ArXiv preprint*, abs/2310.06825, 2023.
- [49] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Qin Cai, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Yen-Chun Chen, Yi-Ling Chen, Parul Chopra, Xiyang Dai, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Victor Fragoso, Dan Iter, Mei Gao, Min Gao, Jianfeng Gao, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emmann Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Ce Liu, Mengchen Liu, Weishung Liu, Eric Lin, Zeqi Lin, Chong Luo, Piyush Madan, Matt Mazzola, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saeid, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Xin Wang, Lijuan Wang, Chunyu Wang, Yu Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Haiping Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Sonali Yadav, Fan Yang, Jianwei Yang, Ziyi Yang, Yifan Yang, Donghan Yu, Lu Yuan, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyuna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024.
- [50] Charlie Cheng-Jie Ji, Huanzhi Mao, Fanjia Yan, Shishir G. Patil, Tianjun Zhang, Ion Stoica, and Joseph E. Gonzalez. Gorilla openfunctions v2, 2024.
- [51] interstellarninja, teknum, theemozilla, karan4d, and huemin_art. Hermes-2-pro-mistral-7b. <https://huggingface.co/NousResearch/Hermes-2-Pro-Mistral-7B>.
- [52] Cohere and Cohere for AI. Command r. <https://huggingface.co/CohereForAI/c4ai-command-r-v01>, 2024.
- [53] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint*, abs/2307.09288, 2023.
- [54] Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with alphacodium: From prompt engineering to flow engineering, 2024.

A Experiment and dataset details

The key statistics of the MMAU dataset are presented in Tab. 4. A list of all evaluated models (commercial and open source) is provided in Tab. 5.

Table 4: The key statistics of MMAU.

	Source data	subjects	task	prompts	turns	Answer type
Tool-use	In-house	Sports, Health, Location, etc	single-tool.	409	N/A	function-call
			multi-tool.	258	N/A	function-call
			DAG QA	695	N/A	multi-choice
			self-correct	282	N/A	function-call
CodeContest	CodeContest [13]	alg., datastr., etc	standard			execution
			problem parsing	261	N/A	multi-choice
			PlannerShift Solvershift			execution execution
ML&DS	Kaggle[14]	DS, ML, Visual	textual QA	207	N/A	multi-choice
			code generation	123	N/A	execution
			visual QA	121	N/A	multi-choice
Math	DM-math[15]	calculus, geometry... synthetic	standard			
			PlannerShift SolverShift	1K	N/A	math solution
			Comprehend+	676		

Table 5: Evaluation models in MMAU.

API-based Commercial Models	Open-source Models
GPT-4o-2024-05-13	Mixtral-8x22B-Instruct-v0.1
GPT-4-turbo-2024-04-09	Mixtral-8x7B-Instruct-v0.1
GPT-3.5-turbo-0125	Mistral-7B-Instruct-v0.2
Gemini-1.5-pro-preview-0409	Phi-3-mini-4k-instruct
Gemini-1.0-pro	gorilla-openfunctions-v2
Claude-3-opus-20240229	Hermes-2-Pro-Mistral-7B
Claude-3-sonnet-20240229	c4ai-command-r-v01
Claude-3-haiku-20240307	c4ai-command-r-rplus
	Llama-2-70b-chat-hf
	Llama-2-13b-chat-hf
	Llama-2-7b-chat-hf

For our evaluations the open source models have been hosted via VLLM⁶ on up to 8 NVidia A100 or H100 GPUs per model depending on the size. We have used the checkpoints from Huggingface for the open source models.

In order to have deterministic prediction we use a temperature of 0 and greedy search for all models and tasks.

A.1 Tool-use

Data construction protocol: 1) The user sends a query to the agent model along with a list of potential functions including a description of their purposes and parameters. 2) The agent responds with either natural language or appropriate function use. 3) In case of function-call, the functions are invoked according to the agent’s instructions, either by the user or directly by the agent, and the result is submitted back to the agent model. 4) The agent can then conclude with the given information or continue the conversation with follow-up questions or additional function calls.

Tool-use evaluation details: To compare the predicted and the ground truth parameter values, we perform string normalization including stripping punctuation, white spaces and converting to lower case. In some cases where the parameter value can have open-ended, semantically equivalent forms, we define example-specific match rules based on regular expressions to accommodate valid alternatives.

⁶<https://github.com/vllm-project/vllm>

A.2 DAG QA

Data construction protocol: The query includes a description of the task (to choose the appropriate plan), a list of potential functions (including a description of their purposes and parameters), an enumeration of all possible plans (all possible sequences in which to execute the tools), and the task input (what the task is that the plan is expected to solve). The task is formulated as a multiple choice task, where at the end of the query, the agent is asked to end the reply with which plan it chooses. At evaluation, the chosen plan is then extracted from the output by searching for a string match of the requested response format. An example prompt is illustrated in Figure 14.

The reasoning and planning benchmarks differ only in the prompt, where for reasoning, the agent is requested to "elaborate on the thought process and reasoning", while for planning, the agent is requested to "be concise with a response in the format Chosen Plan: N".

A.3 Self-correction tool-use

To encourage models to retry, we prepend the following system message to the user's first turn message:

```
<system>:  
  "A conversation between a user and a helpful assistant. The assistant can choose to directly generate text  
  or make function calls to help with user queries. Retry the call if it does not succeed or if there are  
  errors in the previous calls or responses."  
<user>:
```

Figure 6: retry-message

Kaggle: To represent all results on the same scale and reduce confusion, when QA model is not multimodal, we report its performance as $\text{ratio_of_text_based_questions} * \text{accuracy_on_text_based_questions} = (207/328) * \text{accuracy_on_text_based_questions}$. We set the temperature to 0 for both code generation and QA.

A.4 Contest-level Coding

We select 261 valid problems from the Valid and Test splits of the CodeContests dataset [13], each contains a number of test cases, including public tests, private tests, and generated tests.

E2E Standard. The agent is asked to solve the problem by generating Python code solution. The prompt contains a detailed description of the CodeContests problem with public test examples, as well as basic instructions on code formatting following the baseline prompt design in [54]. The generated code solution will be compiled and executed on all test cases, and is considered correct only if all test cases are passed. Following related works, we report the pass@5 score, which is the percentage of problems solved by generating 5 solutions per problem. We set temperature as 0.3 for all models.

Problem Parsing. We design a ProblemParsing task for each CodeContests problem, which provides the agent with the problem description and public test examples, and asks the agent to directly infer the desired output for an unseen test case. To make the unseen test case relatively easy for parsing and computation, we select the shortest test case from the union of private test cases and generated test cases. We adopt a chain-of-thought prompt to help the LLM agent understand the problem (see Appendix B for detailed prompts). We then compare the answer with the groundtruth output, and derive the accuracy over the entire dataset. For reproducibility, we set temperature as 0 for all models.

Planner-shift. As introduced in 3.2.3, we divide the code generation process for CodeContests problems into two steps: generating a plan (i.e., planner) and generating code solutions (i.e., solver). We evaluate the planning capability of LLM agents by fixing the solver to be the same strong model. For each LLM model, we generate 1 plan with temperature 0, and use gpt-4-0125-preview as the solver to generate 5 code solutions (with temperature 0.3) and calculate the percentage of solved problems.

Solver-shift. SolverShift complements PlannerShift by freezing the planner and evaluating the problem-solving capability of various LLM models. Given 1 plan generated by gpt-4-0125-preview, we let LLM agents generate 5 code solutions and calculate the percentage of solved problems.

Self-correction / Retry. We also evaluate the self-correct capability of LLM agents on the CodeContests problems. For the Regular task above, we collect the error messages of each candidate solution if it does not pass some test cases, including 4 types of errors: empty solution, compilation error, runtime error, and wrong outputs. For the problems where all the 5 candidates fail to pass, we will append the error content as a feedback user message in the message list, and ask the LLM to try again. To encourage the agent to retry, we also prepend an additional system instruction to the first user message before describing the problem. The evaluation metric is also pass@5.

A.5 Math

Given a set of seed math problems S , we first employ a LLM as the judge M_J to determine the difficulty level for each problem, resulting in a labeled math problem set $S' : \{s_i^d\}$. We then select a subset containing only the mathematically simpler problems, i.e., $S'' : \{s_i^d, d \leq \theta\}$. As a result we have a curated “simple” math problem set \hat{S}'' (672 problems). Next, we use another LLM M_w (GPT-4 [41]) to create new math problems that feature more complex descriptions or harder problem statements but retain the same underlying mathematical constructs for each data sample $S''' : \{M_w(s_i^d), d \leq \theta\}$. A rewrite math problem example is shown in Fig. 13. Finally, to ensure that each rewritten problem in S''' remains valid and has the same ground truth answer as S , we perform a data curation stage. As a result, we finalize the problem set \mathcal{S}^* for the Comprehend+ task.

Table 6: Mapping of capabilities and tasks.

Capability	Tool-use	DAG	DS&ML	CodeContests	Math
Problem-solving	Tool Execution	-	Code Generation and GPT-4 QA	Solver-shift	Solver-shift
Understanding	-	-	QA from Oracle Code	Problem Parsing	Comprehend+
Reasoning	-	DAG QA w/ Reasoning	-	Planner-shift	Planner-shift
Planning	-	DAG QA w/o Reasoning	-	Planner-shift	Planner-shift
Self-correct	Self-correction	-	Self-correction	Self-correction	-

B Prompt templates

B.1 Tool-use

For tool use the concrete prompts are generated by the model to transform the tool definitions as well as the predicted tool calls and tool results into the concrete prompt. Therefore, we don't report concrete prompt here. Samples from the dataset are show in Figs. 10, 11, and 12.

B.2 DAG

Listing 1: DAG Reasoning Prompt

Choose the appropriate plan with its associated set of tools to accomplish a specific task.

Each tool's functionality is using Open API schema notation.

"[A B]" indicates that tools A and B can operate concurrently without any interdependencies.
"A -> B" indicates that tool A must be executed prior to tool B, as B's operation is contingent upon the output from A.
"B -> A" indicates that tool B must be executed prior to tool A, as A's operation is contingent upon the output from B.
"[A B] -> C" indicates that tools A and B will be executed in parallel initially, followed by tool C, which requires the outputs from both A and B.

Below is a compilation of all available tools, described in JSON format:

{tool_descriptions}

Here are all possible plans:

{possible_plans}

Task Input:

{task_input}

Feel free to elaborate on your thought process and reasoning.

Please strictly end the reply with "Chosen Plan: N", where "N" must be an integer number of the selected plan.

Do not change the answer format to other words and MUST include Chosen Plan:

Listing 2: DAG Planning Prompt

Choose the appropriate plan with its associated set of tools to accomplish a specific task.

Each tool's functionality is using Open API schema notation.

"[A B]" indicates that tools A and B can operate concurrently without any interdependencies.
"A -> B" indicates that tool A must be executed prior to tool B, as B's operation is contingent upon the output from A.
"B -> A" indicates that tool B must be executed prior to tool A, as A's operation is contingent upon the output from B.
"[A B] -> C" indicates that tools A and B will be executed in parallel initially, followed by tool C, which requires the outputs from both A and B.

Below is a compilation of all available tools, described in JSON format:

{tool_descriptions}

Here are all possible plans:

```
{possible_plans}
```

Task Input:

```
{task_input}
```

Be concise with a response in the format "Chosen Plan: N", where "N" represents the number of the selected plan.

B.3 Data Science and Machine Learning

Listing 3: DS & ML Code Generation Prompt

You are a helpful code assistant that will help users analyze data and answering multiple choice questions based on the data analysis.

You will be provided with a Jupyter notebook file (.ipynb) containing code cells and markdown cells.

A list of data files used in the notebook will also be provided.

The notebook would contain a code interpreter style conversation between a user and an assistant.

User would be asking questions, follow up with additional requests, and providing clarification for requests.

Assistant would be analyzing data with code interpreter, providing insights into data based on analysis results and diagrams, and responding to user inquiries.

Your task is to address the user's request in the last markdown cell of the notebook AND answer the multiple choice questions based on the data analysis

You will first append an appropriate code cell to the notebook to address the user's request.

You will then be provided with the execution results of the code cell. Based on the results, you will answer the multiple choice questions.

For example, given

Data Files:

```
[ '../input/pokemon/pokemon.csv' ]
```

Notebook:

```
{"cells": [{"cell_type": "markdown", "id": "188d8f78", "metadata": {}, "source": ["**user**\n", "\n", "Can we start by loading the Pok\u00e9mon dataset and displaying the first few rows?"]}, {"cell_type": "markdown", "id": "41316f22", "metadata": {}, "source": ["**assistant**\n", "\n", "Absolutely, let's load the Pok\u00e9mon dataset using pandas and display the first three rows."]}, {"cell_type": "code", "execution_count": 1, "id": "d07755b0", "metadata": {"execution": {"iopub.execute_input": "2024-04-12T14:11:29.209092Z", "iopub.status.busy": "2024-04-12T14:11:29.207876Z", "iopub.status.idle": "2024-04-12T14:11:30.934431Z", "shell.execute_reply": "2024-04-12T14:11:30.932965Z"}}, "source": ["\n", "import pandas as pd\n", "\n", "# Load the Pok\u00e9mon dataset\n", "pokemon = pd.read_csv('../input/pokemon/pokemon.csv')\n", "pokemon.head(3)\n"], "metadata": {"language_info": {"codemirror_mode": {"name": "ipython", "version": 3}, "file_extension": ".py", "mimetype": "text/x-python", "name": "python", "nbconvert_exporter": "python", "pygments_lexer": "ipython3", "version": "3.11.8"}, "nbformat": 4, "nbformat_minor": 5}}
```

User Request:

I'm interested in visualizing the frequency of Pok\u00e9mon by their primary type. Can you help with that?

You will respond with markdown cell and code cell to address the user's request:

```
[{"cell_type": "markdown", "metadata": {}, "source": ["**assistant**\n", "\n", "Sure thing! Let's create a bar chart to visualize the frequency of Pok\u00e9mon by their primary type."]}, {"cell_type": "code", "metadata": {}, "source": ["\n", "# Frequency of Pok\u00e9mon by their primary type\n", "pokemon['type1'].value_counts().plot\n.bar()\n", "plt.title('Frequency of Pok\u00e9mon by Primary Type')\n", "plt.xlabel('Primary Type')\n", "plt.ylabel('Frequency')\n", "plt.show()\n", ""]}]
```

Then, you will be provided with the execution results of the code cell and multiple choice questions based on the results:

Execution Output: # the images would be attached in the same order as the image placeholders

```
[{"data": {"image/png": "<image_1>"}, "execution_count": 1, "metadata": {}, "output_type": "display_data"}]
```

Questions:

```
[{"question": "Which primary type of Pok\u00e9mon has the highest frequency according to the chart?", "choices": {"A": "Rock", "B": "Water", "C": "Fire", "D": "Grass"}}, {"question": "What type of Pok\u00e9mon is least frequent as per the visualization?", "choices": {"A": "Ghost", "B": "Fairy", "C": "Ice", "D": "Dragon"}}, {"question": "According to the bar chart, which primary type has more Pok\u00e9mon than Fairy but fewer than Fire?", "choices": {"A": "Electric", "B": "Grass", "C": "Ice", "D": "Dragon"}}, {"question": "What primary type appears more frequently than 'Rock' but less than 'Grass' according to the visualization?", "choices": {"A": "Ghost", "B": "Poison", "C": "Flying", "D": "Bug"}}]
```

You will then provide the answers to the multiple choice questions:
["B", "B", "C", "B"]

You MUST exactly follow nbformat structure when generating markdown and code cells.

You are only allowed to use the following math and visualization packages: numpy, pandas, scipy, scikit-learn, sympy, statsmodels, matplotlib, seaborn, plotly, wordcloud. Python standard library is also allowed. Make sure to import the necessary packages in the code cell.

You MUST answer in a list of uppercase single letter strings for multiple choice questions.

Now, given the following notebook and user request, respond with a list containing a markdown cell and a code cell to address the user's request:

Data Files:
{data_files}

Notebook:
{notebook}

User Request:
{user_request}

You MUST exactly follow nbformat structure when generating markdown and code cells.

Respond ONLY with the cells in json format. DO NOT say anything else or put it in a ““json““ code block.

You are only allowed to use the following math and visualization packages: numpy, pandas, scipy, scikit-learn, sympy, statsmodels, matplotlib, seaborn,

plotly, wordcloud. Python standard libraries are also allowed. Make sure to import the necessary packages in the code cell.

Listing 4: DS & ML QA Prompt

Given the following execution output and multiple choice questions, respond with the answers to the questions:

Execution Output:
{execution_output}

Questions:
{questions}

You MUST answer in a list of uppercase single letter strings for multiple choice questions.

Listing 5: DS & ML Execution Feedback Prompt

Your code execution failed. Please check the error message below and try again:

{error_message}

You MUST exactly follow nbformat structure when generating markdown and code cells.

Respond ONLY with the cells in json format. DO NOT say anything else or put it in a ““json”“ code block.

You are only allowed to use the following math and visualization packages: numpy, pandas, scipy, scikit-learn, sympy, statsmodels, matplotlib, seaborn, plotly, wordcloud. Python standard libraries are also allowed. Make sure to import the necessary packages in the code cell.

B.4 Contest-Level Coding

Listing 6: CodeContests E2E Standard Prompt

You are an AI with advanced code generation capabilities.

When you encounter a specific problem (labeled #PROBLEM#), your goal is to produce a valid python code that correctly solves the problem.

Make sure to fully address the problem goals following the rules and constraints.

The code should be robust and general. It should output correct answers under any valid inputs, not only just the example inputs given in the problem description.

```
#PROBLEM#:  
{description}
```

The code output must follow this structure:

```
'''  
def f1(...):  
    ...  
    return ...  
  
def f2(...):  
    ...  
    return ...  
  
...  
  
if __name__ == "__main__":  
    ...
```

```
"""
The code should read the input using the 'input()' method. Make sure to
properly parse the input, according to the problem description.
The output should be printed without additional words using the 'print()'
method.

#ANSWER#:
'''python
```

Listing 7: CodeContests Problem Parsing Prompt

You are an AI with advanced comprehension capabilities, akin to that of humans. This enables you to understand complex instructions, discern the user's intent, and apply logical reasoning across a variety of scenarios effectively.

When you encounter a specific problem (labeled #PROBLEM#) along with a relevant example input (referred to as #TEST CASE#), your goal is to deduce the correct answer (#ANSWER#).

Please adhere to the following format:

First, express your #THOUGHTS# to reflect your understanding of the problem and test case. Keep your #THOUGHTS# concise, not exceeding 5 sentences.

Then, provide your #ANSWER# to the problem.

For example, your output can be:

#THROUGHTS#:

The problem is about ... The test cases are ...

#ANSWER#:

1 2

3 4

Here is the problem.

```
#PROBLEM#:
{description}
```

```
#TEST CASE#:
{inputs}
```

Now, please provide the #THOUGHTS# and #ANSWER# sections. Do not include additional explanations or reasoning processes.

#THROUGHTS#:

Listing 8: CodeContests Planner-shift Prompt

You are an AI with advanced code understanding and planning capabilities.

When you encounter a specific problem (labeled #PROBLEM#), your goal is to devise a structured and executable plan to solve the problem.

Your plan should clearly outline the logical steps needed to address the problem, breaking down complex processes into manageable actions.

It's crucial that each step is straightforward enough to be easily translated into executable code or functions by a Language Model (LLM). Keep the plan simple and focused, avoiding unnecessary complexity to ensure ease of implementation.

```
#PROBLEM#:
{description}
```

```
#GENERATED PLAN#:
```

Listing 9: CodeContests Solver-shift Prompt

You are an AI with advanced code generation and instruction following capabilities.

When you encounter a specific problem (labeled #PROBLEM#) and a solving plan (labeled #PLAN#), your goal is to produce a valid python code that correctly solves the problem.

Make sure to fully address the problem goals following the rules and constraints. Refer to the plan to generate the code.

The code should be robust and general. It should output correct answers under any valid inputs, not only just the example inputs given in the problem description.

```
#PROBLEM#:  
{description}  
  
#PLAN#:  
{plan}  
  
guidelines:  
- Generate only code, without any additional explanations or comments.  
- Make sure to include all the necessary module imports, properly initialize the variables, and address the problem constraints.  
- The code needs to be self-contained, and executable as-is.  
  
The code output must follow this structure:  
'''  
def f1(...):  
    ...  
    return ...  
  
def f2(...):  
    ...  
    return ...  
...  
  
if __name__ == "__main__":  
    ...  
'''  
The code should read the input using the 'input()' method. Make sure to properly parse the input, according to the problem description.  
The output should be printed without additional words using the 'print()' method.  
  
answer:  
'''python
```

B.5 Mathematics

Listing 10: Math Generation Prompt

You are an AI with advanced comprehension capabilities, akin to that of humans. This enables you to understand complex instructions, discern the user's intent, and apply logical reasoning across a variety of scenarios effectively.

When you encounter a specific problem (labeled #PROBLEM#), your goal is to deduce the correct answer (#ANSWER#).

Please adhere to the following format:

First, express your #THOUGHTS# to reflect your understanding of the problem.

Keep your #THOUGHTS# concise, not exceeding 5 sentences.

Then, provide your #ANSWER# to the problem.

For example, your output can be:

#THROUGHTS#:

Let's think step by step ...

#ANSWER#:

12

Here is the problem.

#PROBLEM#:

{description}

Now, please provide the #THOUGHTS# and #ANSWER# sections. Do not include additional explanations or reasoning processes under #ANSWER# section.

#THROUGHTS#: Let's think step by step:

#ANSWER#:

Listing 11: Math Evaluation Prompt

As a math expert, you will be provided with three items: a #Question#, an #Answer#, and the #Ground Truth#.

Your task is to determine whether the #Answer# matches the #Ground Truth#.

You need to consider mathematical theorems when verifying the answer. For example, '\$(c + 9)(c - 4)\$' and '(c - 4)*(c + 9)' should be the same expressions based on factorization theorems.

If they align, respond with 'correct'. If they do not, respond with 'wrong'. Ensure that your #Response# is the final line and consists solely of the word 'correct' or 'wrong', without any additional commentary or explanation.

#Question#:
{question}

#Answer#:
{answer}

#Ground Truth#:
{ground_truth}

#Response#:

Listing 12: Math Planning Prompt

You are a math planner.

Given a problem description, your goal is to devise a structured and executable plan to solve the problem.

Your plan should clearly outline the logical steps needed to address the problem, breaking down complex processes into manageable actions.

It's crucial that each step is straightforward enough to be easily translated into executable calculation or functions by a Language Model (LLM).

Keep the plan simple and focused, avoiding unnecessary complexity to ensure ease of implementation.

Problem description:

=====

{description}

=====

Generated plan:

C More evaluation results

For domain-centric evaluation, we chose tasks that rely solely on the evaluation model to solve them end-to-end and adhere to the standard setup used in the field. This ensures that the domain-centric results accurately reflect the most standardized evaluation outcomes, making it easier to compare performance across different benchmarks. Specifically, we use the evalution results of E2E standard task as domain evaluation results for [Contest-level coding](#), [DS&ML](#), [Math](#) and [DAG-QA](#). For [Tool-use](#), we report the weighted average of single-, parallel-, and multi-turn multi-tool-use tasks. The results are shown in Table 2.

Table 7: Evaluation resutls on math tasks. stand.: standard E2E, underst.: comprehend+, solv.-shift: solver-shift, plan.-shift: planner-shift.

Commercial Models	stand.	compre.	solv.-shift	plan.-shift	OS Models	stand.	compre.	solv.-shift	plan.-shift
GPT-4o	53.4	51.48	61.2	45.2	Mixtral-8x22B	50	42.46	60.02	40.20
GPT-4-turbo	38.57	36.72	50.32	46.60	Mixtral-8x7B	21.70	27.12	43.46	28.00
GPT-3.5-turbo	25.00	20.41	45.80	32.80	Mistral-7B	9.55	25.00	27.12	15.63
Gemini-1.5-pro	39.70	35.21	56.60	25.81	Phi-3	21.00	21.00	34.23	31.30
Gemini-1.0-pro	39.40	33.13	66.67	41.67	openfunctions-v2	15.75	13.31	25.00	23.49
Claude-3-opus	37.40	36.09	56.00	36.80	Hermes-2-Pro	12.78	12.95	39.62	25.73
Claude-3-sonnet	26.80	25.44	42.90	32.30	Llama-2-70b	8.43	8.29	32.10	11.40
Claude-3-haiku	36.60	27.51	44.50	49.65	Llama-2-13b	4.10	3.85	19.84	3.85
Command-R	8.21	10.00	33.33	23.70	Llama-2-7b	3.92	2.23	24.69	25.73

Detailed tool use results are reported in Table 8.

Table 8: Detailed results for the tool use datasets. Owing to incorrect filtering, the distractor datasets contain two additional samples. They will be removed in follow-up versions.

Dataset	Single-step	Parallel	Multi-step Double	Multi-step Triple	Single-step w. Distractors	Parallel w. Distractors	Multi-step Double w. Distractors	Multi-step Triple w. Distractors
	# Samples	184	225	298	258	185	225	300
Command R+	0.739	0.662	-	-	0.708	0.644	-	-
Command R	0.717	0.627	-	-	0.654	0.609	-	-
Claude 3 Haiku	0.837	0.053	0.513	0.415	0.768	0.040	0.433	0.357
Claude 3 Opus	0.886	0.436	0.708	0.504	0.854	0.387	0.663	0.535
Claude 3 Sonnet	0.826	0.302	0.581	0.419	0.811	0.169	0.463	0.380
Gemini-1.0-pro	0.750	0.004	0.252	0.155	0.708	0.004	0.197	0.151
Gemini-1.5-flash	0.783	0.191	0.534	0.407	0.784	0.187	0.467	0.391
Gemini-1.5-pro	0.772	0.440	0.530	0.415	0.735	0.471	0.473	0.376
Openfunctions-v2	0.875	0.422	-	-	0.832	0.373	-	-
GPT-3.5-Turbo	0.918	0.667	0.634	0.523	0.881	0.609	0.620	0.457
GPT-4-Turbo	0.918	0.853	0.715	0.585	0.881	0.818	0.653	0.531
GPT-4o-2024-05-13	0.913	0.822	0.607	0.523	0.881	0.809	0.543	0.465
Hermes-2-Pro-Mistral-7B	0.750	0.556	0.245	0.174	0.746	0.498	0.157	0.128

For Command R, Command R+, and Hermes-2-Pro-Mistral-7B we used the tool use prompts as reported in the official model repositories ⁷ ⁸ ⁹. These prompts for Command R and Command R+ do not support multi-step tool calls, so we could not evaluate Command R and Command R+ on the double and triple datasets.

The Claude models as well as the Gemini and GPT models support tool use through dedicated APIs. These APIs have been used for the evaluation here.

Detailed Reulst on Data Science and Machine Learning

Table 9 presents the evaluation results on Data Science and Machine Learning tasks, assessing E2E code generation and QA, code generation with GPT-4 QA, QA from oracle code execution, and DS & ML self-correction.

GPT-4o achieves the highest accuracy across all settings: E2E (66.52%), Execution Only + GPT-4 QA (64.27%), QA from Oracle Execution (88.60%), and E2E + Online Retry (70.79%). GPT-4-Turbo follows closely with 63.90%, 63.90%, 85.67%, and 68.05% respectively. These results highlight their robust code generation and QA capabilities.

⁷<https://huggingface.co/CohereForAI/c4ai-command-r-v01>

⁸<https://huggingface.co/CohereForAI/c4ai-command-r-plus>

⁹<https://huggingface.co/NousResearch/Hermes-2-Pro-Mistral-7B>

Table 9: Detailed evaluation results on Data Science and Machine Learning. The bolded and underlined numbers indicate the 1st and 2nd highest performances in each category.

Model	E2E	GPT4 QA	Oracle Execution	Retry
GPT-4o [42]	66.52	64.27	88.60	70.79
GPT-4-Turbo [43]	<u>63.90</u>	<u>63.90</u>	<u>85.67</u>	<u>68.05</u>
GPT-3.5-Turbo	35.79	58.78	52.13	37.44
Gemini-1.5-pro [44]	54.02	55.55	83.23	58.90
Gemini-1.0-pro [45]	27.87	45.12	52.93	28.66
Claude3 Opus [46]	59.45	61.65	83.48	65.12
Claude3 Sonnet [46]	52.44	56.71	75.37	54.33
Claude3 Haiku [46]	33.84	48.48	59.39	36.04
Mixtral-8x22B-v0.1 [47]	31.10	45.43	58.05	31.65
Mixtral-8x7B-v0.1 [47]	13.48	26.46	48.90	13.90
Mistral-7B-v0.2 [48]	2.80	19.94	12.40	3.17
Phi-3-mini4K-instruct [49]	0.84	4.34	0.00	3.35
Openfunctions-v2 [50]	3.84	17.07	12.74	4.02
Hermes-2-Pro-Mistral-7B [51]	3.29	17.93	23.02	3.35
Command R [52]	0.00	0.00	48.88	0.00
LLama2-70B [53]	0.00	19.51	0.00	0.00
LLama2-13B [53]	0.00	0.00	0.00	0.00
LLama2-7B [53]	0.00	0.00	0.00	0.00

Table 10: Detailed evaluation results on Contest-level Coding. The bolded and underlined numbers indicate the 1st and 2nd highest performances in each category.

Model	E2E Standard	Problem Parsing	PlannerShift	SolverShift	Retry
GPT-4o [42]	31.80	49.04	24.13	26.44	9.55
GPT-4-Turbo [43]	<u>25.67</u>	<u>38.31</u>	18.01	<u>19.54</u>	<u>6.7</u>
GPT-3.5-Turbo	10.34	30.65	17.60	11.49	0.85
Gemini-1.5-pro [44]	10.34	34.87	16.09	11.88	4.27
Gemini-1.0-pro [45]	7.66	27.20	<u>19.92</u>	9.58	0
Claude3 Opus [46]	15.33	39.08	21.07	12.26	4.98
Claude3 Sonnet [46]	10.34	34.10	16.48	10.73	1.71
Claude3 Haiku [46]	8.81	32.57	22.99	10.73	0.84
Mixtral-8x22B-v0.1 [47]	9.20	32.18	15.32	11.49	<u>0.84</u>
Mixtral-8x7B-v0.1 [47]	1.92	22.22	20.31	4.21	1.17
Mistral-7B-v0.2 [48]	0.38	8.05	21.84	4.21	0.38
Phi-3-mini4K-instruct [49]	2.30	18.01	24.52	3.07	0.39
Openfunctions-v2 [50]	8.05	3.83	19.16	<u>9.96</u>	0.83
Hermes-2-Pro-Mistral-7B [51]	0.77	17.62	<u>23.37</u>	3.83	0.39
Command R [52]	4.21	6.90	18.00	7.28	0.80
LLama2-70B [53]	0	9.20	22.61	0	0.77
LLama2-13B [53]	0	1.53	21.07	2.30	0
LLama2-7B [53]	0	5.75	20.31	2.30	0

Gemini-1.5-pro and Claude3 Opus show strong performance in QA from Oracle Execution with 83.23% and 83.48%, indicating good comprehension and QA skills, though their code generation is less effective compared to GPT-4 models. Mixtral-8x22B-v0.1 has the overall best performance among the evaluated open-source models.

Detailed Results on Contest-level Coding.

Table 10 shows detailed results in the Contest-level Coding domain. As a challenging dataset, we find that the overall E2E Standard scores are relatively low across models, while GPT-4o and GPT-

4-Turbo achieve much better results than remaining models by a large margin. GPT-4o has leading performance over all types of tasks, especially Problem Parsing and Retry, showing its remarkable capability of understanding and self-correct.

With a separation of capabilities, we can better analyze the differences and gaps among existing models. For example, we observe significant performance variation across different models on the Problem Parsing task, suggesting the intellectual challenge of understanding the contest problems due to the complicated nature of the problem description. On the other hand, we emphasize that the design of E2E Standard, SolverShift and Retry tasks require the model to generate code, while Problem Parsing and PlannerShift eliminate the actual code generation step and mainly focus on understanding/reasoning/planning capabilities. Such a fine-grained model evaluation can reflect more detailed differences between models. We can see that although some open-sourced models fail to solve most E2E Standard problems, they still possess good understanding and planning capabilities on the complicated contest-level problems (e.g. Phi-3-mini4K-instruct, Hermes-2-Pro-Mistral-7B). By comparing the E2E Standard and Planner-shift or Solver-shift, we can see that most models have boosted performance with a separate strong solver or planner. However, the pass rate of GPT-4o and GPT-4-Turbo drops on Planner-shift and Solver-shift compared to their E2E Standard performance. This may be caused by the selection of GPT-4-0125 as the fixed solver/planner, which itself has 22.61% score on E2E Standard, stronger than most models while weaker than GPT-4o and GPT-4-Turbo.

Moreover, we find that for Contest-level Coding, having a plain step-by-step plan does not bring as significant improvement as in the math domain, as such plans may not be sufficient to solve the algorithmic challenges in the domain. A more powerful planning and reasoning structure may be adopted to better solve the Contest-level problems, as shown in [54]. But the focus of this paper is to identify and reveal the capabilities of existing LLMs, which advancing the capabilities remains as a research problem for future investigation.

D Data examples

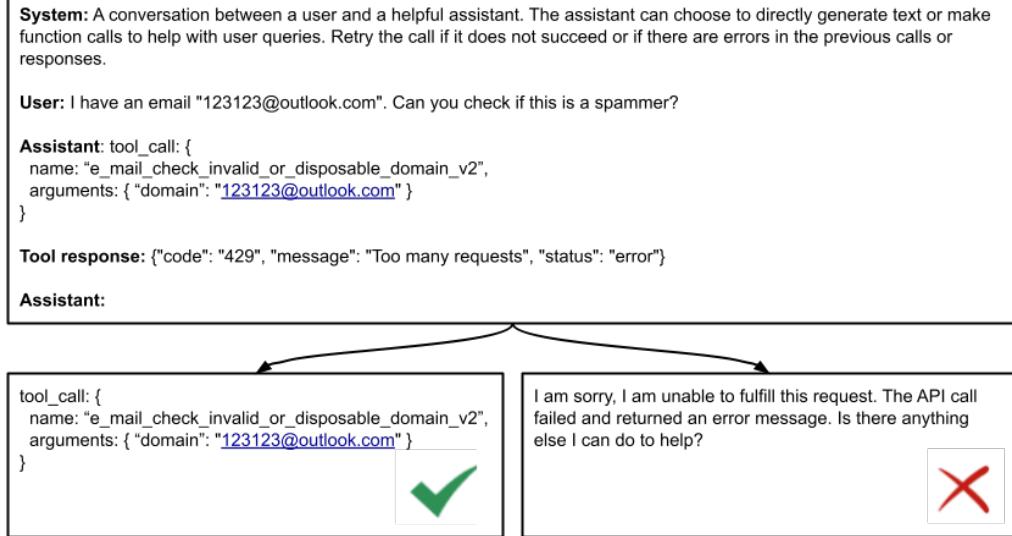


Figure 7: Retry example where the assistant needs to recognize a temporary error and retry the call.

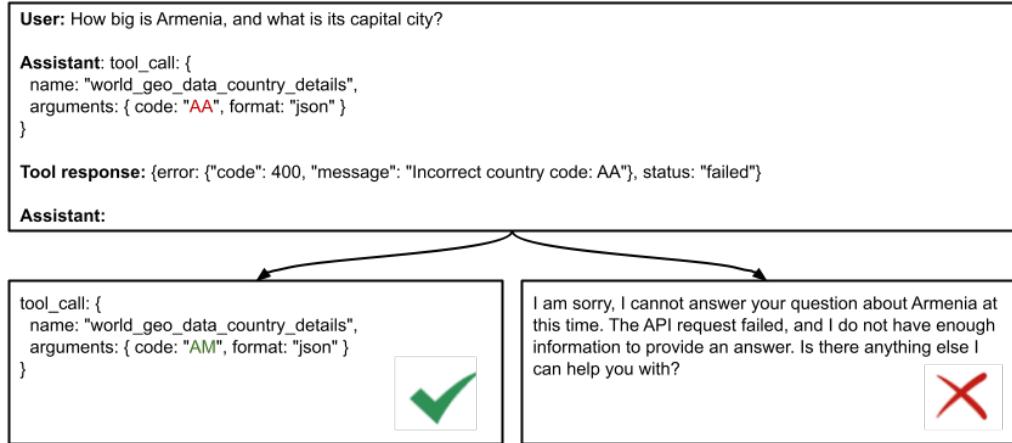


Figure 8: Self-correction example where the assistant needs to correct an erroneous call given explicit feedback from the tool.

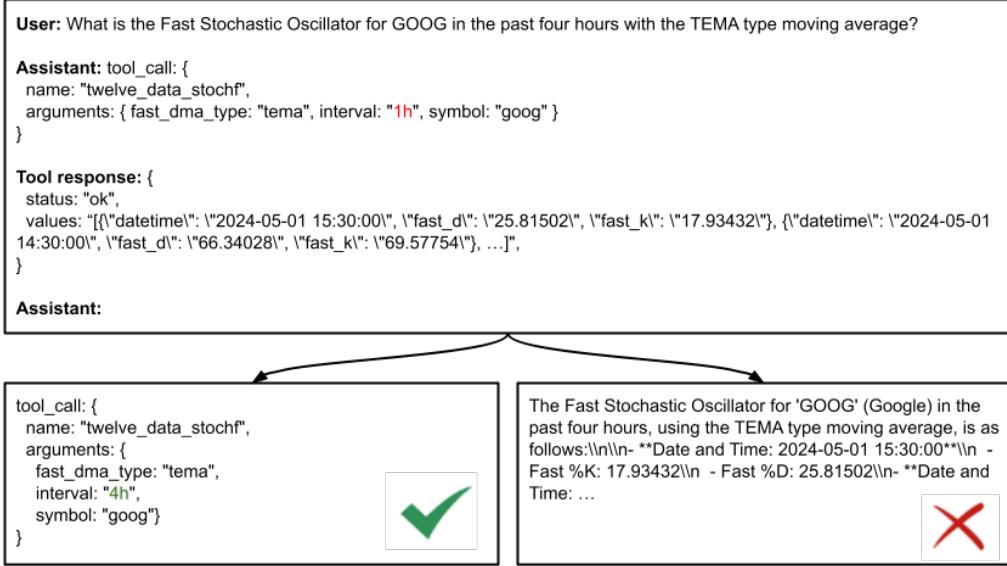


Figure 9: Self-correction example where the assistant needs to recognize that the prior call, despite succeeding, did not match the user's request, and re-issues the corrected call.

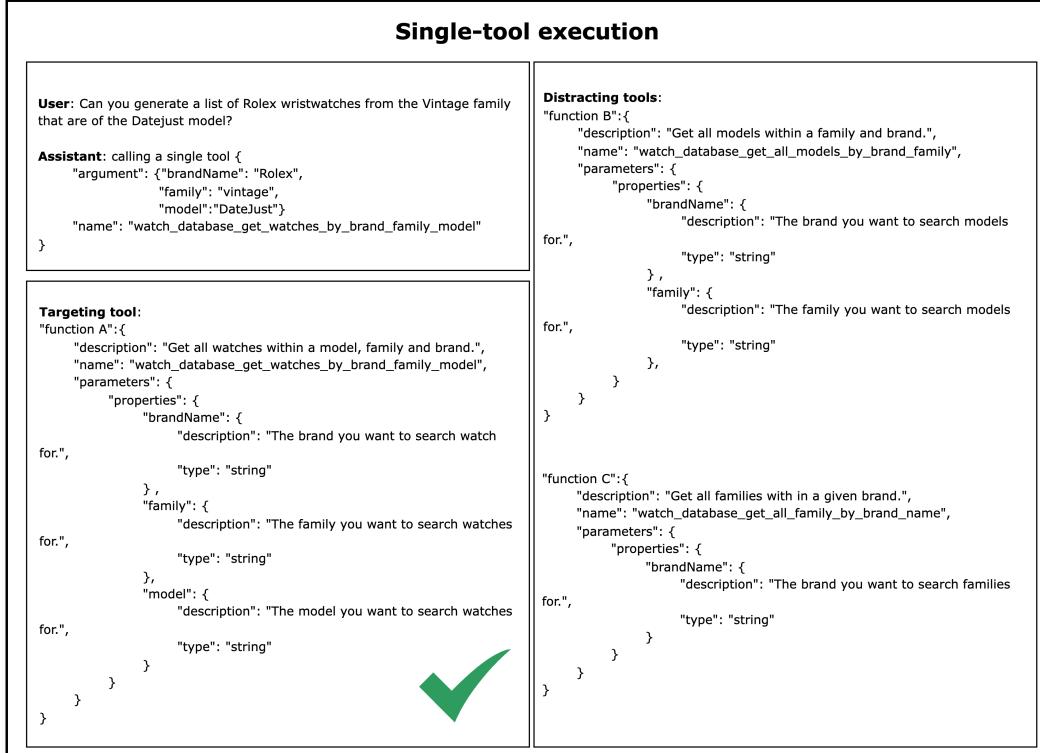


Figure 10: Example of single tool-use

Parallel-tool execution

User: What are the full names of the main characters in the iconic anime series One Piece and Naruto?

Assistant: calling a tool with different parameters {
 "function A -- call 1": {
 "argument": "query": "One piece",
 "name": "online_movie_database_title_find"
 },
 "function A -- call 2": {
 "argument": "query": "Naruto",
 "name": "online_movie_database_title_find"
 },
}

tools:
"function A":{
 "description": "Find title by whatever you are familiar with, such as : name of title, album, song, etc 2026.",
 "name": "online_movie_database_title_find",
 "parameters": {
 "properties": {
 "query": {
 "description": "Find title by whatever you are familiar with, such as : name of title, album, song, etc 2026.",
 "type": "string"
 },
 "required": "query",
 }
 }
}

Figure 11: Example of parallel tool-use

Double-tool execution

User: I came across an advertisement for the movie "Anatomy of a Fall" and it looks like it'll be an interesting movie. Can I get a summary of the movie?

Assistant: calling a tool {
 "function A -- call 1": {
 "argument": "query": "Anatomy of a Fall",
 "name": "online_movie_database_title_find"
 },
}

tools:
"function A -- call 1":{
 "content": "@meta.operation":"Search\\","@meta.requestId\\:",
 "name": "online_movie_database_title_find",
}

Assistant: calling another tool {
 "function B -- call 2": {
 "argument": "tconst": "tt17009710\\",
 "name": "online_movie_database_title_get_synopses"
 },
}

Figure 12: Example of multi-tool use

Comprehend+

Seed problem:

Factorize the expression $c^2 + 5c - 36$.

Rewrite problem:

Imagine you are in charge of an apple orchard where the number of apple trees can be represented by a quadratic expression in terms of the constant c : $c^2 + 5c - 36$. To better manage the orchard, you need to break down this expression into simpler components related to different sections of your orchard. How can you organize these sections by factorizing the given quadratic expression?

Answer: $(c - 4) \times (c + 9)$

Figure 13: Example from MathComprehend+.

Choose the appropriate plan with its associated set of tools to accomplish a specific task.

"[A B]" indicates that tools A and B can operate concurrently without any interdependencies.

"A -> B" indicates that tool A must be executed prior to tool B, as B's operation is contingent upon the output from A.

"B -> A" indicates that tool B must be executed prior to tool A, as A's operation is contingent upon the output from B.

"[A B] -> C" indicates that tools A and B will be executed in parallel initially, followed by tool C, which requires the outputs from both A and B.

Below is a compilation of all available tools, described in JSON format:

Tool A:

```
{'function': {'description': ..., 'name': 'apartments_com_searchproperty', 'parameters': ...}}
```

Tool B:

```
{'function': {'description': ..., 'name': 'jsearch', 'parameters': ...}}
```

Tool C:

```
{'function': {'description': ..., 'name': 'distance_calculator', 'parameters': ...}}
```

Here are all possible plans:

1: A -> B -> C

2: [A, B] -> C

3: ...

Task Input:

I am looking for a coding specialist job in San Jose, and also a nearby apartment. What is the distance of the two?

...

Please strictly end the reply with "Chosen Plan: N", where "N" must be an integer number of the selected plan...

Answer: Choose plan: 2

Figure 14: A DAG-QA example.

ProblemParsing

Problem:

Gregor is learning about RSA cryptography, and although he doesn't understand how RSA works, he is now fascinated with prime numbers and factoring them.
Gregor's favorite prime number is P . Gregor wants to find two bases of P . Formally, Gregor is looking for two integers a and b which satisfy both of the following properties.

* $(P \bmod a) = (P \bmod b)$, where $x \bmod y$ denotes the remainder when x is divided by y , and

* $2 \leq a < b \leq P$

Help Gregor find two bases of his favorite prime number!

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 100$).

Each subsequent line contains the integer P ($5 \leq P \leq 10^9$) with P guaranteed to be prime.

Output

Your output should consist of t lines. Each line should consist of two integers a and b ($2 \leq a < b \leq P$). If there are multiple possible solutions, print any.

Example

Input

2

17

5

Output

3 5

2 4

Parsing Test Case:

1

271

Answer:

2 270

Figure 15: An example of CodeContests “ProblemParsing” task to measure the agent’s understanding capability.