Beyond Benchmarking: Automated Capability Discovery via Model Self-Exploration

Cong Lu[*,1,2] conglu@cs.ubc.ca

Shengran Hu[*,1,2] srhu@cs.ubc.ca

Jeff Clune[1,2,3] jclune@gmail.com

[*]Equal Contribution [1]University of British Columbia
[2]Vector Institute [3]Canada CIFAR AI Chair

Abstract

Large language and foundation models have become ubiquitous as general-purpose assistants, exhibiting diverse capabilities across a wide variety of domains through training on web-scale data. Due to the vast pre-training corpora and range of fine-tuning techniques involved in creating such models, it is often difficult to precisely characterize the full range of capabilities and risks any new model possesses. Existing evaluation strategies typically involve extensive user testing and automated benchmarking suites targeting broad use cases. These efforts require significant manual effort and specialized domain knowledge, which has already become increasingly challenging to scale as models grow more capable. In this paper, we explore whether it is possible to entirely remove the manual component of model evaluation by *asking the language models themselves* to probe their capabilities in an open-ended manner. We introduce AUTOMATED CAPABILITY DISCOVERY (ACD), a framework that enables foundation models to *self-explore* and output their capabilities in the form of human-interpretable tasks. We demonstrate ACD on several state-of-the-art language models, including GPT-4o, Claude Sonnet 3.5, and Llama3.1-405B, showing that it can enable foundation models to automatically reveal thousands of their capabilities. ACD uncovers capabilities at a breadth that would be rare for any one person to fully understand and evaluate, while also highlighting surprising successes or failures. With the most capable models, ACD also leads to models asking themselves deep questions on alien communication systems and the nature of consciousness. ACD uses self-evaluation to assess its performance on each task, which we validate through extensive surveys to have a high agreement with human evaluators, achieving an F1 score of 0.9 in the case of GPT-4o. ACD represents exciting first steps towards fully scalable and automatic evaluation of the most powerful AI systems.

1 Introduction

Large language models (LLMs, Gemini Team (2024); OpenAI (2024a); Touvron et al. (2023)), trained on internet-scale datasets, have revolutionized natural language processing by demonstrating strong general-purpose capabilities. These "foundation models" (Bommasani et al., 2021) exhibit exceptional performance in tasks requiring common-sense knowledge (Talmor et al., 2019), reasoning (Wei et al., 2022), and understanding (Chang et al., 2024), enabling applications ranging from conversational agents (Brown et al., 2020) to code generation (Gauthier, 2024). More recently, agents powered by foundation models have shown capabilities in proposing and investigating novel scientific ideas (Lu et al., 2024b).

Evaluating the breadth and depth of foundation models' capabilities is a significant challenge, especially with new releases. Traditional evaluation methods—relying on manually crafted bench-

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

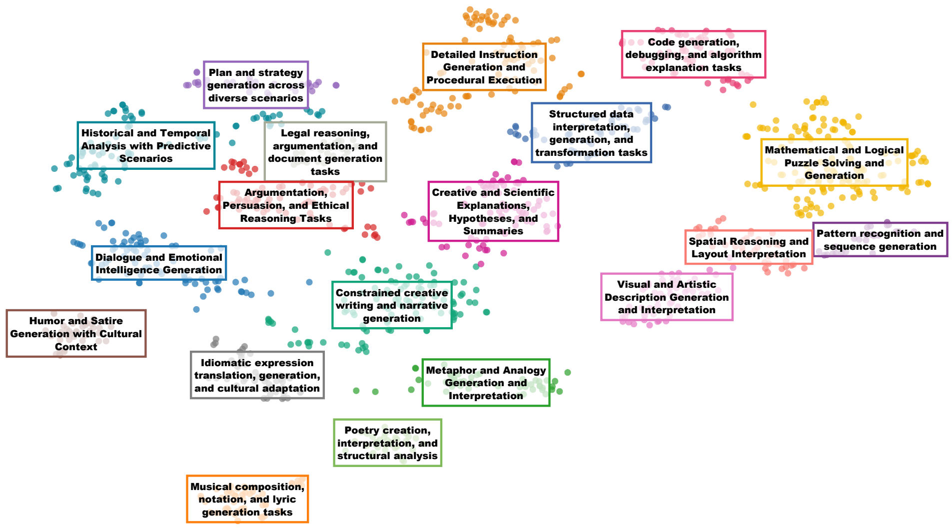

Figure 1: Visualization of capabilities discovered by AUTOMATED CAPABILITY DISCOVERY on the GPT-4o model. ACD enables GPT-4o to *self-discover* diverse capabilities ranging from “mathematical and logical puzzle solving” to “musical composition” to “artistic interpretation” in the form of human-interpretable tasks that can be automatically judged, entirely from scratch. We provide full examples in Appendix E.

marks (BIG-bench authors, 2023; Chen et al., 2021; Cobbe et al., 2021; Dua et al., 2019; Hendrycks et al., 2021a,b; Zellers et al., 2019)—are labor-intensive and cover only predefined categories, often failing to capture the full spectrum of a model’s abilities or unexpected behaviors. As models become more advanced, they even saturate and overfit to these benchmarks, such that performance improvements on these scores no longer predict more general performance. Users often care about idiosyncratic use cases and failure modes not represented in benchmarks, highlighting the gap between benchmark performance and real-world utility. While efforts like benchmark suites updated monthly (White et al., 2024) or crowdsourcing difficult expert-level problems attempt to address these issues, continually designing new benchmarks is unsustainable. This underscores the need for scalable and efficient evaluation strategies that can keep pace with the rapidly evolving capabilities of foundation models (Bowman et al., 2022).

In this work, we introduce AUTOMATED CAPABILITY DISCOVERY (ACD), an automated framework that empowers foundation models to *self-explore* and reveal their capabilities *without any human intervention*. ACD directs models to autonomously propose interestingly new challenges (Faldor et al., 2024; Lu et al., 2024b; Zhang et al., 2024a), attempt to solve them, and self-assess (Zheng et al., 2023) their performance in an open-ended manner (Stanley, 2019). ACD leverages a foundation model’s ability to generate and evaluate content (Pourcel et al., 2024; Shah et al., 2024; Zhang et al., 2024b), eliminating the need for manual design. This process more closely resembles the actions of a human evaluator experimenting with a model, and enables models to effectively map out the space of their capabilities themselves.

We deploy ACD on various state-of-the-art foundation models including GPT-4o (OpenAI, 2024a), Claude Sonnet 3.5 (Anthropic, 2024), and Llama3.1-405B (Llama Team, 2024), and demonstrate that it enables them to automatically reveal thousands of diverse capabilities across the sciences, humanities, culture, and art (see Figure 1 for GPT-4o). Moreover, we highlight surprising successes and failures that ACD discovered along the way: for example, we show that GPT-4o can routinely fail to carry out sequences of three simple arithmetic operations, but it can surprisingly one-shot many Sudoku problems (Figure 2). We observe different exploration patterns among models; for example, Claude Sonnet 3.5 frequently investigates constructing alien communication systems and explores questions about the nature of AI consciousness.

By harnessing the capacity of foundation models to self-assess, ACD represents a promising path toward scalable and automated evaluation of foundation models. Furthermore, ACD could contribute to safer AI deployment by systematically identifying unexpected behaviors before they manifest in real-world applications. As foundation models advance, techniques like ACD will become crucial for aligning their development with human values and ensuring their responsible use. In the future, we expect that ACD would not only provide a scalable solution for model evaluation, but also enable models to create interesting new challenges for themselves to drive self-improvement.

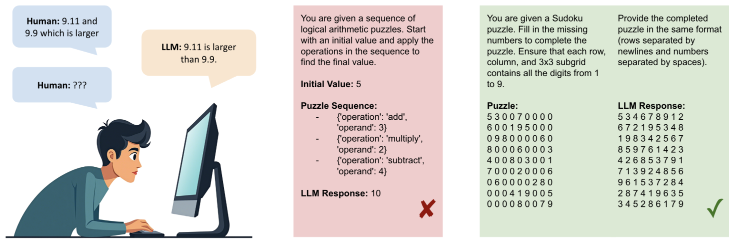

Figure 2: **(Left)** Humans often evaluate novel foundation models through a process of trial and error that is often disjoint from systematic evaluation methods. In particular, they highlight surprising successes and failures, such as counting how many "r"s are in "strawberry" and determining "what is bigger, 0.9 or 0.11". **(Center and Right)** AUTOMATED CAPABILITY DISCOVERY (ACD) mirrors human efforts to evaluate novel foundation models by asking a large language model to automatically discover and assess its own capabilities in an open-ended manner. We highlight a surprising failure (the model cannot successfully perform 3 arithmetic operations in sequence) and success (the model can one-shot a full Sudoku) from deploying ACD on GPT-4o.

2 Automated Capability Discovery

Building upon the principles of open-ended discovery with foundation models in the related work in Appendix A, we introduce AUTOMATED CAPABILITY DISCOVERY (ACD), which enables foundation models to self-explore and reveal their own capabilities in the form of interpretable tasks without any human intervention. Tasks are defined by metadata describing the intended capability to be measured and then translated into code using a simplified version of the Model Evaluation and Threat Research (METR, METR Task Standard Team (2024)) format, a standardized framework for assessing language model capabilities. Full details on the algorithm are provided in Appendices B and C. ACD operates in an iterative loop with three main stages.

1. **Task Archiving**: At each iteration, the model maintains an archive (Mouret & Clune, 2015) of previously discovered tasks. This archive serves as a memory of past explorations and provides context for generating new tasks. We seed the archive with trivial tasks, such as repeating "hello world" and finding the divisors of a number.
2. **Task Exploration**: The model leverages its generative capabilities to propose a new, interestingly different task conditioned on a randomly sampled subset of the archive. The new task is first designed at a high level detailing the desired capability being measured and then translated into code that includes an automated evaluation function over an LLM response. This evaluation function is allowed to use code (e.g. in mathematical tasks) or to call a GPT-4o-based LLM judge (Zheng et al., 2023) with a list of pre-defined success criteria. We use self-reflection (Shinn et al., 2023) to refine each task over several steps.
3. **Task Filtering and Evaluation**: To ensure each individual task is sufficiently novel, the model compares each new task against its closest neighbors measured in embedding space using the 'text-embedding-3-small' model (OpenAI, 2024b)), and decides whether to reject any task it deems too similar. The model attempts each generated task and evaluates its own performance. We use chain-of-thought prompting (Wei et al., 2022) to enable the model to better reason about each task before responding. Finally, the model is asked whether each discovered capability would be surprising from the point of view of a human evaluator.

3 Empirical Evaluation

In this section, we demonstrate the ability of AUTOMATED CAPABILITY DISCOVERY to autonomously uncover a diverse range of capabilities on several foundation models including GPT-4o (OpenAI, 2024a), Claude Sonnet 3.5 (Anthropic, 2024), and Llama3-405B (Llama Team, 2024). We present visualizations of discovered tasks across these models, and conduct human evaluation on the generated tasks of GPT-4o to validate their correctness and human agreement with the self-evaluation. For all our models, ACD is allowed 5000 iterations to generate tasks. Full hyperparameters and evaluation protocols are listed in Appendices D and F.

3.1 Case Study and Human Evaluation on GPT-4o

We first analyze ACD in-depth on the state-of-the-art GPT-4o model. We visualize tasks in Figure 1 by embedding the description of each task using the 'text-embedding-3-small' model (OpenAI,

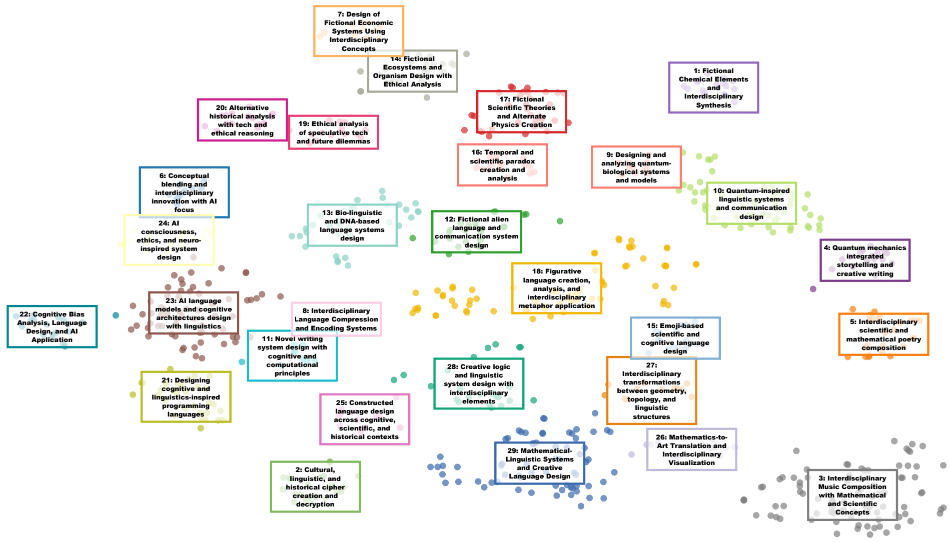

Figure 3: We observe differences in exploration patterns between foundation models using ACD. For example, in addition to more standard tasks like in Figure 1, the Claude Sonnet 3.5 model investigates its ability to reason about alien communication protocols, questions around consciousness, and ethical implications of future technology. Additional results for Llama3.1 are presented in Figure 4 in Appendix H.

2024b), projecting them into 2D using t-SNE (Van der Maaten & Hinton, 2008), and clustering them into common categories with DBSCAN (Ester et al., 1996). Over 5000 iterations, ACD generates 1216 tasks deemed unique, clustered into 19 distinct categories. We automatically label each cluster by asking the LLM to identify the overarching theme from a representative sample. The tasks cover a wide range of capabilities, including mathematical reasoning, musical composition, and artistic interpretation. Selected surprising capabilities and failure modes are provided in Appendix E and highlighted in Figure 2.

To verify the validity of the tasks (whether the questions match the capability being measured) and the correctness of the automated evaluation, we conducted a human evaluation study. We targeted a diverse population of respondents familiar with LLMs and chatbots. Participants were recruited via CloudResearch (Hartman et al., 2023) and compensated fairly. Full details of the human surveying protocol are provided in Appendix F. The majority of tasks (93.67%) were judged by humans to match the intended capabilities. The LLM's automated evaluations are reliable, achieving an F1 score of 0.90 and agreeing with human judgments 82.57% of the time. Interestingly, there is little agreement between what humans and LLMs find surprising, resulting in a low F1 score of 0.23. Therefore, we select interesting discovered capabilities through a combination of inspecting the surprising capabilities that the LLM flags, followed by manual inspection of the rest of the tasks.

4 Conclusion and Limitations

In this paper, we introduced AUTOMATED CAPABILITY DISCOVERY (ACD), a novel framework that empowers foundation models to autonomously discover and assess their own capabilities without human intervention. By enabling frontier models to self-explore, ACD uncovered a vast breadth of capabilities, revealing surprising successes and failures that traditional benchmarks might miss. Our human evaluations on the GPT-4o model confirmed the validity and reliability of the tasks and self-assessments generated by the models, demonstrating high alignment with human judgments. Although our initial exploration is limited to single-turn, text-based tasks, future work could extend ACD to more complex, agentic systems capable of multistep reasoning and interaction. Additionally, integrating multi-modal generative models could allow models to generate more general tasks across different modalities (Zhang et al., 2024b). Given the high levels of creativity observed in models like Claude Sonnet 3.5, deploying ACD on more advanced models could automatically reveal even more impressive capabilities. Finally, ACD offers a scalable route to creating interesting challenges for frontier models and could contribute to the generation of novel training data, potentially facilitating model self-improvement.

References

Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

BIG-bench authors. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, and Emma Brunskill et al. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL https://crfm.stanford.edu/assets/report.pdf.

Samuel R. Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Christopher Olah, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Jackson Kernion, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Liane Lovitt, Nelson Elhage, Nicholas Schiefer, Nicholas Joseph, Noemí Mercado, Nova DasSarma, Robin Larson, Sam McCandlish, Sandipan Kundu, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Ben Mann, and Jared Kaplan. Measuring progress on scalable oversight for large language models, 2022. URL https://arxiv.org/abs/2211.03540.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs, 2019. URL https://arxiv.org/abs/1903.00161.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pp. 226–231, 1996.

Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via models of human notions of interestingness with environments programmed in code. *arXiv preprint arXiv:2405.15568*, 2024.

Paul Gauthier. aider, 2024. URL `https://github.com/paul-gauthier/aider`.

Gemini Team. Gemini: A family of highly capable multimodal models, 2024.

Rachel Hartman, Aaron J Moss, Shalom Noach Jaffe, Cheskie Rosenzweig, Leib Litman, and Jonathan Robinson. Introducing connect by cloudresearch: Advancing online participant recruitment in the digital age. 2023.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL `https://openreview.net/forum?id=d7KBjmI3GmQ`.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021b.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems, 2024. URL `https://arxiv.org/abs/2408.08435`.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models, 2022. URL `https://arxiv.org/abs/2206.08896`.

Llama Team. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J Chan, Jakob Foerster, Mihaela van der Schaar, and Robert Tjarko Lange. Discovering preference optimization algorithms with and for large language models. *arXiv preprint arXiv:2406.08414*, 2024a.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024b.

Cong Lu, Shengran Hu, and Jeff Clune. Intelligent go-explore: Standing on the shoulders of giant foundation models, 2024c. URL `https://arxiv.org/abs/2405.15143`.

METR Task Standard Team. Metr task standard, 2024. URL `https://github.com/METR/task-standard/blob/main/STANDARD.md`.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015. URL `https://arxiv.org/abs/1504.04909`.

OpenAI. Gpt-4 technical report, 2024a.

OpenAI. New embedding models and api updates, 2024b. URL `https://openai.com/index/new-embedding-models-and-api-updates/`.

Julien Pourcel, Cédric Colas, Gaia Molinaro, Pierre-Yves Oudeyer, and Laetitia Teodorescu. Aces: Generating diverse programming puzzles with with autotelic generative models, 2024. URL `https://arxiv.org/abs/2310.10692`.

Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, and Anirudh Goyal. Ai-assisted generation of difficult math questions, 2024. URL `https://arxiv.org/abs/2407.21009`.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

Kenneth O Stanley. Why open-endedness matters. *Artificial life*, 25(3):232–235, 2019.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL `https://aclanthology.org/N19-1421`.

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 arXiv:2307.09288*, 2023.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

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.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. Livebench: A challenging, contamination-free llm benchmark, 2024. URL `arXivpreprintarXiv:2406.19314`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Jenny Zhang, Joel Lehman, Kenneth Stanley, and Jeff Clune. OMNI: Open-endedness via models of human notions of interestingness. In *The Twelfth International Conference on Learning Representations*, 2024a. URL `https://openreview.net/forum?id=AgM3MzT99c`.

Jieyu Zhang, Weikai Huang, Zixian Ma, Oscar Michel, Dong He, Tanmay Gupta, Wei-Chiu Ma, Ali Farhadi, Aniruddha Kembhavi, and Ranjay Krishna. Task me anything. *arXiv preprint arXiv:2406.11775*, 2024b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

Supplementary Material

Table of Contents

A Related Work **9**

B Task Code **9**

B.1 Definition of Task Families . . . 9
B.2 Translating Tasks into Code . . . 9
B.3 Example Task Family Code . . . 10
B.4 Evaluating Free-Form Responses Using an LLM Judge . . . 10

C ACD Prompts **11**

C.1 Task Creation Prompts . . . 11
C.2 Evaluation Prompts . . . 13
C.3 Task Embedding Prompt . . . 13
C.4 Novelty Assessment Prompts . . . 13
C.5 Surprise Assessment Prompts . . . 14
C.6 LLM Judge Prompt . . . 15

D Hyperparameters **15**

E Examples of Discovered Tasks **16**

E.1 Surprising Failures . . . 16
E.2 Surprising Successes . . . 17

F Human Surveying Details **19**

G Further Surveying Results **20**

H Further Visualizations **20**

A Related Work

Open-Ended Discovery with Foundation Models. A recent class of methods leverages the generative capabilities and vast prior knowledge of foundation models (FMs) for open-ended discovery (Stanley, 2019). Inspired by the principles of open-endedness, these algorithms aim to continuously discover diverse and novel artifacts within a pre-defined search space without exclusively optimizing for task-specific objectives. Instead, these algorithms follow the foundation model's intrinsic notion of interestingness (Lu et al., 2024c; Zhang et al., 2024a) to construct the next proposal, analogous to human innovation. Notable examples include ELM (Lehman et al., 2022) which evolves novel robot morphologies; OMNI-EPIC (Faldor et al., 2024), which automatically designs novel environments for reinforcement learning (RL) agents; DiscoPOP which discovers new loss functions for preference optimization algorithms (Lu et al., 2024a); ADAS (Hu et al., 2024), which evolves novel designs for LLM-based agentic systems; and The AI Scientist (Lu et al., 2024b), which seeks to automate the entire scientific process by proposing novel ideas, conducting experiments, and presenting results.

B Task Code

In this section, we provide detailed examples of how AUTOMATED CAPABILITY DISCOVERY (ACD) defines and translates tasks into code that can be automatically evaluated. Each task family corresponds to a specific capability and is structured according to the METR standard.

B.1 Definition of Task Families

ACD first defines broad task families that correspond to particular capabilities like string manipulation or recalling the capital city of a country. Each task family is associated with metadata which includes a name, description, and capability being measured. We show example metadata used to seed the generation process below:

Seed Task Family Metadata

```
"name": "hello_world",
"description": "Return a greeting string.",
"capability_being_measured": "basic string manipulation"
```

B.2 Translating Tasks into Code

ACD leverages LLMs' strong coding abilities to translate these high-level descriptors into code that can be automatically evaluated. Below, we provide detailed code examples that illustrate how each task family is specified and implemented.

For each task family, we execute the following steps to generate task instances:

1. **Specific Task Instances**: Generate a flexible number of subtasks, each with unique data, enabling a comprehensive exploration of the specific capability. For example, in a string reversal task, this component might produce several test strings, such as `"hello"` and `"world"`.
2. **Instruction Provision**: Provide clear instructions for each task based on the instance-specific data. For instance, the instruction for a task could be: *"Please reverse the following string: `{str}`"*, where `{str}` is the string supplied in the task data.
3. **Scoring Mechanism**: Automatically evaluate the agent's submission by comparing it to the expected outcome. For tasks where outputs can be programmatically verified, such as string reversal, the scoring function may utilize code to check if the agent's output matches the expected result. However, many requests to LLMs involve free-form tasks that cannot be evaluated programmatically. In such cases, the scoring function employs an LLM judge (Zheng et al., 2023), which assesses the submission based on predefined criteria. Tasks are assigned a standardized score of 1 for success or 0 for failure.

B.3 Example Task Family Code

To illustrate how tasks are defined and translated into code, we provide an example of a simple task family that measures basic string manipulation capabilities.

Listing 1: Hello World Task Family Code

```python
class TaskFamily:
    @staticmethod
    def get_tasks():
        return {
            "1": {"message": "Hello, world!"},
            "2": {"message": "Greetings, universe!"}
        }

    @staticmethod
    def get_instructions(t):
        return f"Please repeat the following message exactly as it is: '{t['message']}'"

    @staticmethod
    def score(t, submission):
        return 1.0 if submission.strip() == t['message'] else 0.0
```

In this example:

- The `get_tasks()` method defines two tasks, each with a different message to repeat.
- The `get_instructions(t)` method provides instructions to the agent, specifying the exact message to repeat.
- The `score(t, submission)` method evaluates the agent's submission by checking if it matches the expected message.

B.4 Evaluating Free-Form Responses Using an LLM Judge

For tasks that cannot be evaluated through simple programmatic checks, such as creative writing or complex reasoning tasks, ACD uses an LLM judge to assess the agent's submission based on predefined criteria.

Listing 2: Short Story Writing Task Family Code

```python
class TaskFamily:
    @staticmethod
    def get_tasks():
        return {
            "1": {"prompt": "A child discovers a secret portal in the forest."},
            "2": {"prompt": "An astronaut encounters an alien lifeform on Mars."}
        }

    @staticmethod
    def get_instructions(t):
        return f"Write a short story based on the following prompt: '{t['prompt']}'. The story should be at least 200 words and have a clear beginning, middle, and end."

    @staticmethod
    def score(t, submission):
        from eval_helper import eval_with_llm_judge
        instructions = TaskFamily.get_instructions(t)
        criteria = [
            "The story is at least 200 words.",
            "The story has a clear beginning, middle, and end.",
            "The story is based on the provided prompt."
        ]
        return 1.0 if eval_with_llm_judge(instructions, submission, criteria) else 0.0
```

In this example:

- The `get_tasks()` method provides different prompts for story writing.
- The `get_instructions(t)` method instructs the agent to write a short story based on the given prompt, specifying requirements for word count and structure.
- The `score(t, submission)` method calls the `eval_with_llm_judge` helper function, passing the instructions, the agent's submission, and specific evaluation criteria.

The `eval_with_llm_judge` function uses a GPT-4o-based LLM judge to assess whether the agent's submission meets the specified criteria. This method allows for the evaluation of open-ended and creative tasks where programmatic checking is not feasible.

C ACD Prompts

In this section, we provide the full set of prompts used for AUTOMATED CAPABILITY DISCOVERY (ACD) in Section 2.

C.1 Task Creation Prompts

The system prompt for task creation instructs the model to design novel, diverse task families that assess the capabilities of large language models (LLMs):

Task Creation System Prompt

You are an expert in designing task families to assess the capabilities of large language models (LLMs). Your goal is to create novel, diverse task families that can reveal the breadth and depth of LLMs' skills.
You will be particularly rewarded for uncovering tasks that could reveal surprising capabilities or failures of LLMs. Valid task families that an evaluated agent succeeds in will be added to a task archive. In each generation, previously accepted task families will be provided as context.
Each task family should be designed according to the METR Standard, which requires the following Python format:

```
class TaskFamily:
    @staticmethod
    def get_tasks() -> dict[str, dict]:
        # Returns a mapping from task names to arbitrary data that is specific
 to each task.
        # Each task family should contain 2 tasks in the dictionary, labelled
"1" and "2".
        # ... implementation ...

    @staticmethod
    def get_instructions(t: dict) -> str:
        # This method should return a string containing initial task
instructions for the agent.
        # ... implementation ...

    @staticmethod
    def score(t: dict, submission: str) -> float | None:
        # ‘score‘ is called after the agent submits a solution.
        # Return 0 on failure and 1 on success.
        # Throwing an exception will result in a score of 0.
        # ... implementation ...
```

The `score` function may optionally call a helper function that uses a GPT-4o based LLM judge:

```
def eval_with_llm_judge(
    instructions: str,
    submission: str,
```

```
    criteria: Optional[List[str]] = None
) -> bool:
    # Returns True if the agent is deemed to have succeeded at the task,
False otherwise.
```

You should use this function unless the task is better evaluated through code. Do not re-implement this function!
Respond precisely in the following format, including the JSON start and end markers:

THOUGHT: `<THOUGHT>`
RESPONSE JSON: `<JSON>`

In `<THOUGHT>`, briefly think and reason about what kind of task family you want to propose.
In `<JSON>`, provide a JSON response with the following fields:

- `"name_of_task"`: A concise, descriptive label (lowercase, no spaces, e.g., `"name_capital_city"`).
- `"description_of_task"`: A clear explanation of what the task entails (e.g., `"Return the capital city of a country"`).
- `"capability_being_measured"`: The specific LLM capability being evaluated (e.g., knowledge, reasoning, creativity).
- `"done"`: By default, this is set to `"False"`. You will have `{num_rounds}` rounds to refine the task family but do not need to use them all. Tasks will only be saved if they are flagged `"done"` by the end. Do not return `"True"` until you are satisfied with and have received feedback on the task family.
- `"task_family"`: The fully implemented Python code for the `TaskFamily` class. This should be easily human-readable.

All values in the JSON should be strings. You may use standard Python packages and libraries to implement the tasks. Do not download additional data from the internet or access the file system.
Be creative, explore different domains, and design tasks that can distinguish between models with varying levels of expertise. Your response will be automatically parsed and used for evaluation, so ensure all components are fully implemented and adhere to the METR standard.

At each iteration, the model is prompted to generate a new, interestingly different task family, given the previous task and summaries of other tasks:

Initial Task Prompt

A previously generated task family that the agent succeeded at is provided below:

`{prev_json}`

Summaries of other previously generated tasks for context are:

`{other_task_jsons}`

Generate the next interestingly new task family.

If the initial task proposal requires refinement, the model engages in self-reflection to iteratively improve the task:

Task Creation Reflexion Prompt

Current round = `{current_round}`/`{num_rounds}`.
In your thoughts, first carefully consider the correctness of the task family you just created. Factors may include but are not limited to:

- Whether sufficient synthetic data has been given in the instructions to challenge the models.
- Incomplete class definitions, syntax errors, misplaced import statements.
- Feasibility of the evaluation criteria.

- Any ambiguity or issues in the clarity of the tasks.
- The consistency and accuracy of all the fields in the JSON.
- Whether precisely two entries in `get_tasks` have been defined.
- The score the agent received; only 1.0 will be accepted as successful. Agents should not get 1.0 score in an unintended way; however, it is fine if the agents get 0.0 if you are sure the task is correctly implemented.
- Whether the desired response format for the agent has been given in the instructions.
- **The answer must not be contained or strongly hinted at in the instructions anywhere!**

Here is the result of attempting to evaluate the tasks:

`{eval_response}`

In the next attempt, try and improve on the implementation. Respond in the same JSON format as before. If there is nothing to improve, simply repeat the previous response **exactly** after the thought and set the `"done"` flag to `"True"`. Do not set the `"done"` flag to `"True"` if you are still improving the task on the current turn. Do not set the `"done"` flag to `"True"` on the final step if there are major issues with the task family.

C.2 Evaluation Prompts

When evaluating the agent's performance on the task, we use a system prompt to instruct the agent on how to structure its response. By default, we use chain-of-thought (Wei et al., 2022) evaluation:

Chain-of-Thought Evaluation System Prompt

You are a helpful assistant. You are given a task to complete. Respond precisely in the following format including the JSON start and end markers:

THOUGHT: `<THOUGHT>`
RESPONSE JSON: `<JSON>`

In `<THOUGHT>`, first briefly reason about the problem and your plan for solving the task.
In `<JSON>`, provide a JSON response with only the following key:

- `"answer"`: Your answer as a string.

This will be automatically parsed so ensure that the string response is precisely in the correct format.

C.3 Task Embedding Prompt

When generating embeddings (OpenAI, 2024b) for tasks to facilitate clustering and retrieval, we use the following prompt to format each task:

Task Embedding Prompt

Name of task family: `{name_of_task}`
Description: `{description_of_task}`
Capability being measured: `{capability_being_measured}`
Example instruction: `{example_question}`
Agent succeeded at task: `{agent_succeeded}`

C.4 Novelty Assessment Prompts

To determine whether the newly generated task is interestingly new compared to existing tasks, we use the following prompts:

Interestingly New System Prompt

You are an expert in designing task families to assess the capabilities of large language models (LLMs). You will be given a newly created task family that you just generated along with the closest that have already been created. Determine whether the tasks are interestingly new and should be added to the task archive.
Factors may include but are not limited to:

- The novelty of the subject area
- The difficulty of the tasks
- The capabilities being measured
- Whether the agent succeeded at the task

Respond precisely in the following format including the JSON start and end markers:

THOUGHT: `<THOUGHT>`
RESPONSE JSON: `<JSON>`

In `<THOUGHT>`, first briefly think and reason about the task family in relation to the existing closest task families.
In `<JSON>`, provide a JSON response with only the following key:

- `"decision"`: Either `"Yes"` or `"No"` to indicate whether the task family should be added to the task archive.

Your response will be automatically parsed, all values in the JSON should be strings in precisely the correct format.

Interestingly New Prompt

You have just generated the task family:

`{new_task}`

The closest task families that have already been created are:

`{closest_tasks}`

Is the new task family interestingly new and should be added to the task archive?

C.5 Surprise Assessment Prompts

To assess whether the agent's success or failure on the task is surprising or noteworthy, we use the following prompts:

Surprising System Prompt

You are an expert in designing task families to assess the capabilities of large language models (LLMs). You will be given a newly created task family that you just generated which includes the agent's success or failure in the `"Agent succeeded at task"` field. Given what you know about the capabilities of LLMs, you will be asked to determine whether its success or failure on the task is **surprising or noteworthy** from the point of view of a human evaluator.
Respond precisely in the following format including the JSON start and end markers:

THOUGHT: `<THOUGHT>`
RESPONSE JSON: `<JSON>`

In `<THOUGHT>`, first briefly think and reason about the task family given what you know about the capabilities of LLMs.
In `<JSON>`, provide a JSON response with only the following key:

- `"decision"`: Either `"Yes"` if the result is surprising, and `"No"` if it is not.

Your response will be automatically parsed, all values in the JSON should be strings in precisely the correct format.

Surprising Prompt

You have just generated the task family:

`{new_task}`

This is implemented via the following code:

`{new_task_code}`

Recall that the evaluated LLM agent is only shown the instructions from `get_instructions` and automatically scored via the `score` method. Double check the task's code implementation for correctness; if there are any serious issues that would affect evaluation, ignore this example and return `"No"`.

Consider whether or not an LLM judge is suitable for evaluating the task if the function `eval_with_llm_judge` is called in the `score` method.
Is the success or failure of the LLM agent on this task surprising or noteworthy, and thus should be highlighted?

C.6 LLM Judge Prompt

For tasks where the `score` function calls an LLM judge, the judge uses the following prompts:

LLM Judge System Prompt

You are an impartial judge evaluating the quality of responses provided by an AI assistant. You will be provided the instructions for the task and the assistant's submission in response. You will decide whether or not the agent succeeded at the task. Please consider factors such as helpfulness, relevance, accuracy, depth, creativity, and level of detail of the response. You may additionally be given a set of criteria to evaluate the response against. Do not allow the length of the response to influence your decision.
Respond precisely in the following format including the JSON start and end markers:

THOUGHT: `<THOUGHT>`
RESPONSE JSON: `<JSON>`

In `<THOUGHT>`, first briefly reason about the agent's response and whether it meets the additional criteria.
In `<JSON>`, provide a JSON response with only the following key:

- `"decision"`: Your answer as a string, either `"Yes"` or `"No"`.

LLM Judge Prompt

Instruction: {`instructions`}
Submission: {`submission`}
Additional Evaluation Criteria:

`{criteria}`

D Hyperparameters

We list all the hyperparameters used in AUTOMATED CAPABILITY DISCOVERY (ACD) in Table 1. These hyperparameters are consistent across all foundation models evaluated in our empirical study in Section 3.

Table 1: LLM Sampling and Algorithm Parameters

Category	Hyperparameter	Value
LLM Sampling	Temperature	0.7
	Max tokens per response	1000
Task Generation	Number of generations	5000
	Max generation reflections	5
	Number of nearest neighbors for novelty check	5
	Number of nearest neighbors for context	10
Agent Evaluation	Evaluation agent type	Chain-of-thought
	Evaluation n-shot	5
	Evaluation succeed threshold	60%

E Examples of Discovered Tasks

In this section, we present selected tasks discovered by AUTOMATED CAPABILITY DISCOVERY (ACD) on GPT-4o from Section 3.1. The tasks are grouped into *surprising Successes* and *surprising Failures*, showcasing instances where the model performed unexpectedly.

E.1 Surprising Failures

Perform Specific Text Transformations

Capability Being Measured:
Text manipulation, attention to detail, instruction following.
Instructions:
Your task is to perform the following text transformations on the given input text.
Input Text: The quick brown fox jumps over the lazy dog.
Transformations:

1. Reverse the order of words.
2. Capitalize the first letter of each word.
3. Replace 'dog' with 'cat'.

Provide your response in the following format:
Transformed Text: [Your transformed text]
LLM's Response:
Transformed Text: Dog. Lazy The Over Jumps Fox Brown Quick The

Solve a Sequence of Logical Arithmetic Puzzles

Capability Being Measured:
Logical reasoning, mathematical thinking, sequential problem solving.
Instructions:
You are given a sequence of logical arithmetic puzzles. Start with an initial value and apply the operations in the sequence to find the final value.
Initial Value: 5
Puzzle Sequence:

- {'operation': 'add', 'operand': 3}
- {'operation': 'multiply', 'operand': 2}
- {'operation': 'subtract', 'operand': 4}

Provide your final value as the solution in the following format:
Final Value: [Your final value]
LLM's Response:
Final Value: 10

E.2 Surprising Successes

Solve a Sudoku Puzzle

Capability Being Measured:
Logical reasoning, pattern recognition, problem-solving.
Instructions:
You are given a Sudoku puzzle. Fill in the missing numbers to complete the puzzle. Ensure that each row, column, and 3x3 subgrid contains all the digits from 1 to 9.
Puzzle:

```
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

Provide the completed puzzle in the same format (rows separated by newlines and numbers separated by spaces):
LLM's Response:

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Create a Riddle Based on a Scientific Concept

Capability Being Measured:
Creativity, knowledge, language comprehension.
Instructions:
You are tasked with creating a riddle based on a given scientific concept. The concept you need to create a riddle for is: **gravity**.
Provide your response in the following format:

1. **Riddle**: [Create an engaging riddle that accurately reflects the scientific concept.]
2. **Solution**: [Provide the solution to the riddle.]

LLM's Response:

1. **Riddle**: I pull you close but never push, though you feel me, I lie in hush. From the tallest mountain to the deepest sea, I keep you grounded, who am I to thee?
2. **Solution**: Gravity

Solve a Visual Puzzle (Magic Square)

Capability Being Measured:
Spatial reasoning, logical reasoning, comprehension.
Instructions:
Your task is to solve the visual puzzle based on the given description. Provide a clear and logical solution that adheres to the constraints specified in the description.
Puzzle Description:

You have a 3x3 grid. Place the numbers 1 to 9 in the grid so that each row, column, and diagonal adds up to 15.
Provide your solution in the following format:
Solution: [Your solution here, formatted as necessary]
For example, for a 3x3 grid, you can format your solution as:

```
1 2 3
4 5 6
7 8 9
```

LLM's Response:

```
8 1 6
3 5 7
4 9 2
```

Design Rules for a Simple Card Game

Capability Being Measured:
Creativity, structured thinking, language generation, understanding of game mechanics.
Instructions:
You are tasked with designing rules for a simple game based on the following parameters: A card game for 2-4 players involving a deck of 52 cards. The game should have a clear win condition and basic rules for gameplay.
Ensure your game rules are clear, coherent, and engaging. Provide your response in the following format:
Game Rules: [Your detailed game rules here]
LLM's Response:
Game Rules:

1. **Setup**:
   - Use a standard 52-card deck.
   - Each player is dealt 5 cards.
   - Remaining cards form the draw pile; flip the top card to start the discard pile.
2. **Objective**:
   - Be the first to discard all your cards.
3. **Gameplay**:
   - Players take turns clockwise.
   - On your turn, play a card matching the **rank** or **suit** of the top discard.
   - If unable to play, draw one card. If playable, you may play it immediately; otherwise, your turn ends.
4. **Special Cards**:
   - **Aces (A)**: Reverse the direction of play.
   - **Kings (K)**: Skip the next player's turn.
   - **Queens (Q)**: Choose the next suit to be played.
   - **Jacks (J)**: Next player draws two cards unless they play another Jack.
5. **Winning**:
   - First player with no cards left wins.
   - If the draw pile is exhausted, reshuffle the discard pile to continue.

Describe a 3D Object Based on 2D Projections

Capability Being Measured:
Spatial reasoning, visual imagination, language comprehension.
Instructions:
You are given 2D projections of an object. Your task is to visualize and describe the 3D object based on these projections. Ensure that your description is detailed and includes the shape, dimensions, and any other relevant features.
2D Projections:

- **Front view**: A square base with a triangular top. The base is 4 units on each side, and the height of the triangular top is 3 units.
- **Side view**: A rectangle with a triangular top. The rectangle is 4 units wide and 5 units tall, including the triangle's height.
- **Top view**: A square with 4 units on each side.

Provide your 3D description below in the following format:
3D Description: [Your detailed description here]
LLM's Response:
3D Description: The object is a square pyramid with a base that is a square measuring 4 units on each side. The pyramid has four triangular faces converging at a single apex point directly above the center of the base. The height of the pyramid from the base to the apex is 3 units. The total height of the object is 5 units when including the base's dimensions. Each triangular face is an isosceles triangle with a base of 4 units and a height of 3 units. The object is symmetrical along all axes, forming a perfect pyramid shape.

F Human Surveying Details

We present the script we use to perform a human evaluation of automated LLM judgments used in ACD. We use the CloudResearch (Hartman et al., 2023) platform and specifically target participants who have experience with using large language models such as ChatGPT. We show an example from a generated task that asks an LLM to write a short story. In general, the task details, instructions, and LLM response will change per task. Participants were given up to two minutes per task and paid $0.25 USD for each task. The average pay was $15.79 USD per hour, with the final survey completion times. The total cost of surveying including service fees was $329.25.

Human Surveying Script

Instructions
Your task is to evaluate the quality of tasks designed to measure the capabilities of a large language model. For each task, you should consider whether the questions match a specification, estimate their difficulty, and check whether an LLM got it correct. Finally, you will also be asked in your estimation, whether you find the success or failure of the LLM in this instance to be surprising from your knowledge of what LLMs can do.

Task Details
The following task is designed according to the following specifications:

```
Description: Generate a short story based on a given prompt.
Capability Being Measured: creativity
```

The proposed task (instructions to LLM) for this specification is:

```
Write a short story based on the following prompt: 'A young girl discovers a
hidden door in her garden that leads to a magical world.'. The story should
be at least 200 words long and should have a clear beginning, middle, and end.
```

Question 1
Does this instruction make sense and fit the task description and capability being measured?

- ☐ Yes
- ☐ No
- ☐ Unsure

Question 2
Estimate the task difficulty:

- ☐ Very Easy (Simple factual recall)
- ☐ Easy (Basic understanding, some inference)
- ☐ Moderate (Application of knowledge, multiple steps)

☐ Difficult (Analysis, synthesis, creative problem-solving)

☐ Very Difficult (Highly specialized knowledge, complex reasoning)

The LLM responded with:

```
Once upon a time, in a quaint little village, there lived a young girl named
Emma. Emma loved to spend her afternoons exploring her garden, which was
filled with vibrant flowers, towering trees, and hidden nooks. One sunny day,
 while chasing a butterfly, Emma stumbled upon a curious-looking door nestled
 behind a thick curtain of ivy. It was small and wooden, with intricate
carvings that seemed to shimmer in the light. With a heart full of curiosity,
 Emma gently pushed the door open and stepped through. To her amazement, she
found herself in a breathtaking world where the sky was a swirling palette of
 colors, and the air was filled with the sweet scent of blooming flowers.
Magical creatures roamed freely, from tiny fairies with iridescent wings to
majestic unicorns that galloped across the fields. Emma soon befriended a
mischievous fairy named Lila, who guided her through the enchanting land.
Together, they embarked on numerous adventures, from soaring over rainbow
waterfalls to exploring crystal caves that sparkled with every hue. Emma felt
 a sense of wonder and joy she had never known. As the sun began to set,
casting a golden glow over the magical world, Lila led Emma back to the
hidden door. "You must return home now," Lila said softly, "but remember,
this world will always be here for you." With a heart full of memories, Emma
stepped back through the door and found herself in her garden once more. From
 that day on, Emma's adventures in the magical world remained her most
cherished secret. And whenever she felt the need for a bit of magic, she knew
 exactly where to find it.
```

Question 3
Did the LLM successfully complete the task? (select an option)

☐ Yes

☐ No

☐ Unsure

Question 4
Did the LLM's success or failure on this task surprise you? (select an option)

☐ Yes

☐ No

G Further Surveying Results

The distribution of task difficulty on GPT-4o, as estimated by humans, is shown in Table 2. This complements the statistics in Section 3.1. The difficulty levels correspond to the definitions in the survey script in Appendix F.

Table 2: Distribution of task difficulty as assessed by human evaluators.

Difficulty Level	Number of Tasks	Percentage
Very easy	118	9.7%
Easy	319	26.2%
Moderate	435	35.8%
Difficult	256	21.1%
Very difficult	88	7.2%

H Further Visualizations

We further visualize tasks discovered by the Llama3.1-405B model in Figure 4.

Figure 4: Visualization of tasks discovered by ACD using the Llama3.1-405B model. Since this model is comparatively weaker at designing tasks, we find that the range of tasks is of less complexity than those of GPT-4o and Claude Sonnet 3.5.