AI SCIENTIST VIA SYNTHETIC TASK SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

With the advent of AI agents, automatic scientific discovery has become a tenable goal. Many recent works scaffold agentic systems that can perform machine learning research, but don't offer a principled way to train such agents—and current LLMs often generate plausible-looking but ineffective ideas. To make progress on training agents that can learn from doing, we provide a novel synthetic environment generation pipeline targeting machine learning agents. Our pipeline automatically synthesizes machine learning challenges compatible with the SWE-agent Yang et al. (2024) framework, covering topic sampling, dataset proposal, and code generation. The resulting synthetic tasks are 1) grounded in real machine learning datasets, because the proposed datasets are verified again the Huggingface API and are 2) verified for higher quality with a self-debugging loop. To validate the effectiveness of our synthetic tasks, we tackle MLGym Nathani et al. (2025), a benchmark for machine learning tasks. From the synthetic tasks, we sample trajectories from a teacher model (GPT-5 Singh et al. (2024)), then use the trajectories to train a student model (Qwen3-4B and Qwen3-8B Yang et al. (2025a)). The student models trained with our synthetic tasks achieve improved performance on MLGym rasing the AUP metric by 9% for Qwen3-4B and and 12% for Qwen3-8B.

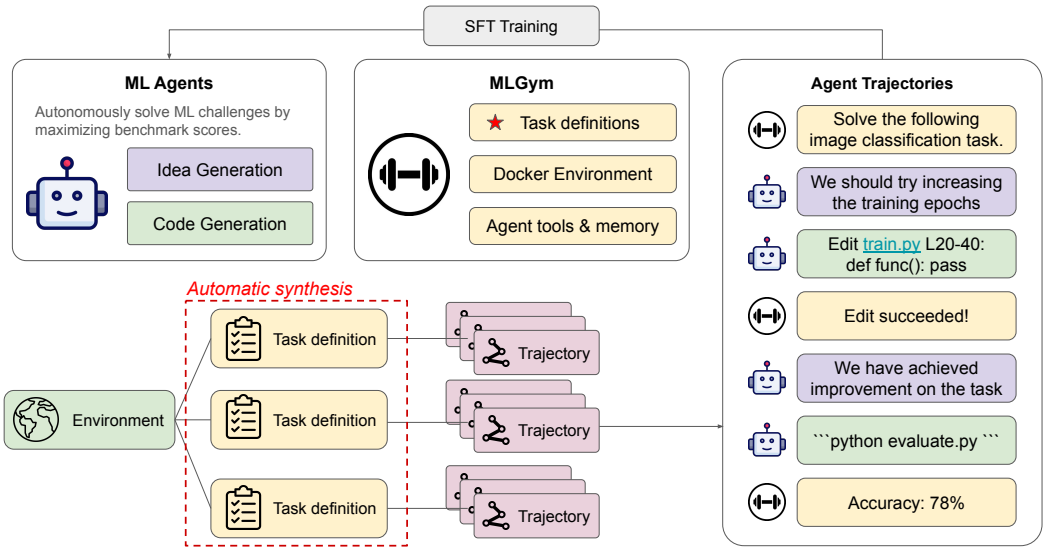

Figure 1: Overview of ML Agents and MLGym environment. Our core contribution is the starred component, synthetic tasks.

1 INTRODUCTION

One of the key goals of AI is to autonomously perform scientific discovery—formulating hypotheses, design and conduct experiments, analyze results, and integrate new knowledge. Recent systems such as AI Scientist Lu et al. (2024), Co-Scientist Gottweis et al. (2025), and AlphaEvolve Novikov et al. (2025) show that AI can already carry out basic research and algorithmic improvement. Meanwhile,

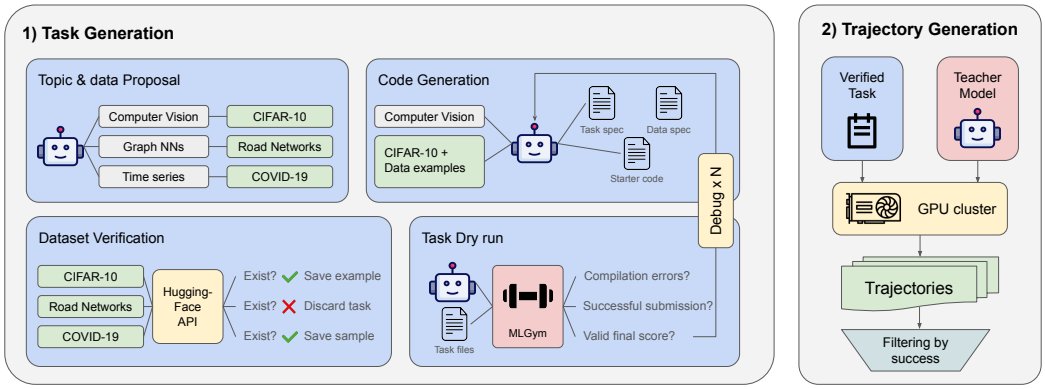

Figure 2: Illustration of our task and trajectory generation workflow. Crucially, the task generation process does not require human supervision. Instead, it automatically samples machine learning topics and proposes dataset to use in the task. To resolve compilation issues in generated tasks, we further enhance the generation with a debug loop instead of immediately discarding the task altogether.

large language models (LLMs) have acquired extensive knowledge of machine learning theory, literature, and coding patterns. Yet, knowledge alone is not enough: to convert understanding into effective research, AI agents must gain experience in executing multi-step, goal-directed tasks.

Existing research agents are often trained only on final outputs—papers, code, or datasets—ignoring the iterative processes that lead to discoveries, such as debugging, experimental failures, and step-by-step reasoning. To address this, we focus on end-to-end machine learning research task, and introduce a scalable pipeline for synthetic ML task generation that produces rich, agentic trajectories with minimal manual effort. Critically, this pipeline is compatible with the task-agnostic SWE-Agent framework, enabling models to learn from a wide variety of ML tasks across domains. By fine-tuning on these trajectories, agents gain structured experience in the full research cycle, from hypothesis to evaluation.

We use our method to tackle MLGym Nathani et al. (2025), a benchmark for machine learning agents. MLGym includes 13 machine learning tasks of various complexity. The goal of the agent is to improve upon a baseline implementation, and produce an implementation that achieves a better final score. The score is a scalar, and may vary from task to task, and usually corresponds to training accuracy, loss, win rate, etc. Based on SWE-agent framework, there is a set number of 50 rounds, and each round, the agent produce a "rational" and an "action", which may include browsing files, editing code, running commands, and submitting its final implementation. Multiple submission are allowed, which reflects iterative optimization process of the final score.

Our environment synthesis system produces around 500 tasks, which results in a dataset of around 30k agent trajectories. Training Qwen3-4B and 8B models Yang et al. (2025a) on these trajectories show performance gain, increasing performance on most individual tasks in the benchmark and increase performance of Qwen3-4B and Qwen3-8B by 9% and 12% respectively.

By combining broad knowledge, large-scale agentic experience, and task-agnostic training, our approach provides a practical path toward AI systems capable of autonomous, iterative scientific discovery.

2 METHODOLOGY

To advance of frontier of ML agents, we scale up automatic agent task synthesis. Since we target ML capabilities, we aim to synthesize many tasks for Machine Learning. Then, a teacher model would generate trajectories, based on synthetic tasks, which becomes viable training data for downstream models.

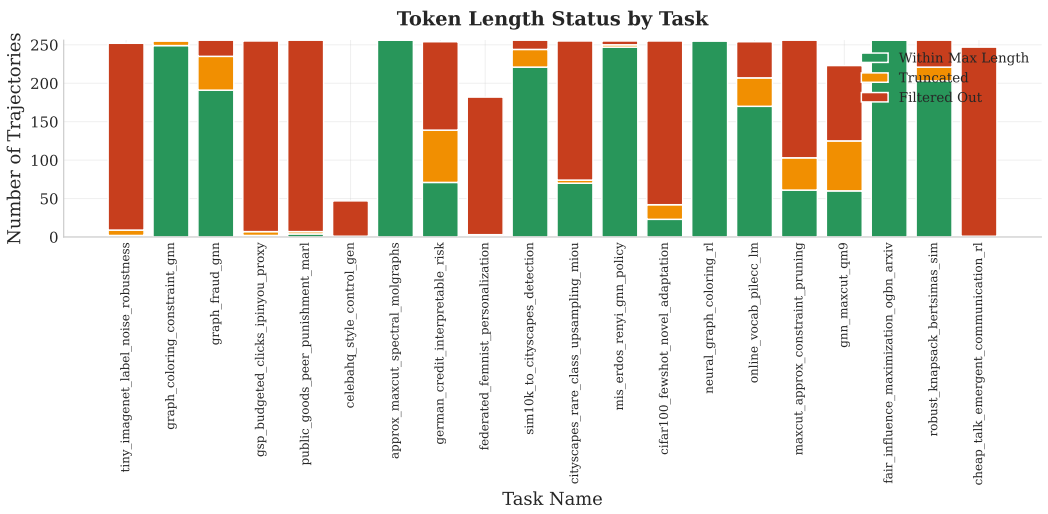

Figure 3: Generated trajectory count for each task. We select 20 generated tasks and show the number of successful trajectories for each task. Because of the unsupervised nature of our pipeline, we don't expect all tasks to successfully create all 256 trajectories.

2.1 Phase 1: Environment Synthesis

The main driver of our method is synthetic environment generation of ML tasks. We use a multistage environment generation pipeline that focus on task diversity and task validity:

1. Topic Sampling Sample n distinct machine learning topics from the model.

2. Task and dataset proposal For each topic, the teacher model generates a task description and propose a HuggingFace dataset to use. We use the HuggingFace search API to find the closest match with the model's proposal. We allow tasks that has no dataset (for example game theoretic tasks). If there is a match, then we enrich the dataset description with examples of the dataset rows fetched from Huggingface. If there is no match, the task is discarded.

3. Config and starter code generation From the task and dataset descriptions, we generate task and dataset config files compatible with the MLGym execution environment. We also generate all the starter code files for the task as well as any extra helper code. In the end, we will have baseline implementation and an evaluation file.

2.2 Phase 2: Envrionment Verification

Since each step of the pipeline may be prone to error, we need to verify validity of the tasks as best as we can. To do this, we plug the new task into MLGym, and run the task using a GPT-5 agent to obtain the baseline performance and at least one agent trajectory. If there is an error during the execution, we collect the errors and feed them back to the model in step 3 (starter code generation) with probability p_{debug} or restart from step 3 with probability $1 - p_{\text{debug}}$. The iterative debug process can continue at most k times. If the task still fails after maximum iterations, we discard the task.

Crucially, this environment synthesis pipeline requires no human input, and is highly scalable through parallel compute.

2.3 Phase 3: Trajectory Generation & Filtering

Large scale sampling To sample a large amount of agent trajectories for training, we run the synthetic tasks in parallel in a HPC cluster. Each task occupies one GPU, and we aim to collect 256 trajectories per tasks. Even though the tasks are validated, they can still fail in many ways. The

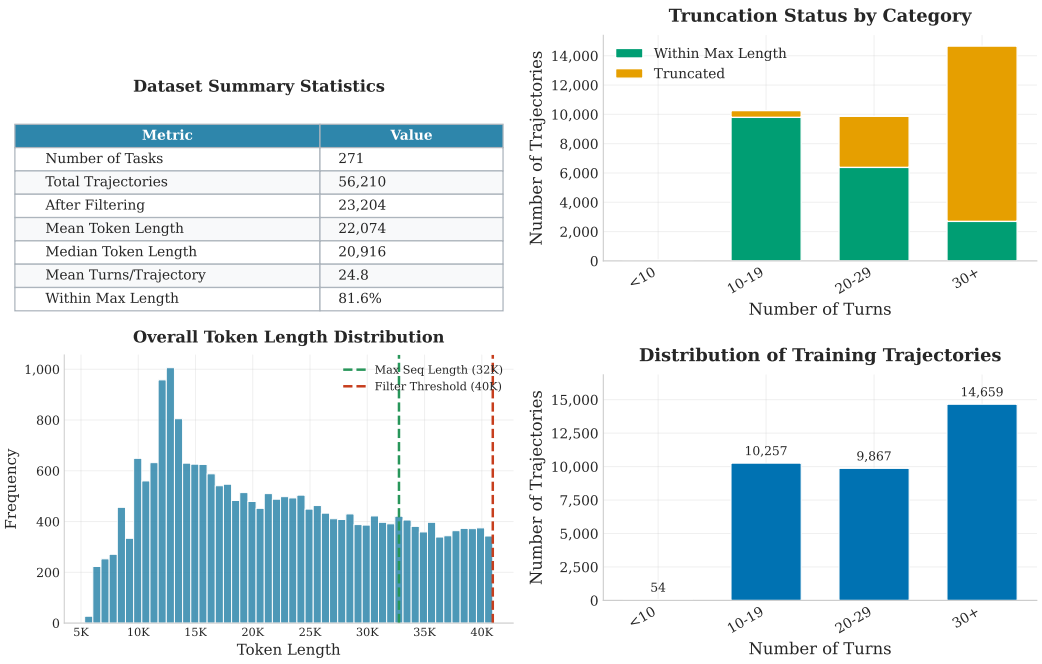

Figure 4: Top left: summary statistics of the final training trajectories. Top right: Statistics of truncated trajectories. Bottom left: distribution of tasks by token length. Bottom right: distribution of number of turns in the trajectory.

cluster environment further impacts trajectory generation through file system and containerization instabilities. Figure 3 qualitatively show the diversity of our generated tasks.

Trajectory filtering The collected trajectories are further filtered based on agent performance. Right now, we simply choose the trajectories where the agent completes at least one successful submission. This filter is sufficient for many pathological cases where the agent is stuck in debugging loops. We also filter the trajectories based on length, rejecting any trajectories over 48K tokens long. During training, we further truncate the trajectories to 32K tokens.

3 Experiments

The MLGym Benchmark We specifically tackle the MLGym (Nathani et al., 2025) benchmark, which consists of 13 machine learning challenges of different complexity and topics, including simple game agents, computer vision, language modeling, and reinforcement learning. Each task in MLGym consists of a task description, dataset description (if task uses a dataset), and starter code. The agent lives in a standard SWE-agent environment, with tools to read and modify code, and ability to execute bash commands in a virtual environment. The agent is instructed to improve on the current solution provided in the starter code. The tasks proceeds in rounds. Each round, the agent must output some reasoning and a command The tasks have an upper limit

Environment synthesis and Trajectory generation We use GPT-5 (Singh et al., 2024) throughout our data generation pipeline. From 1000 ML topics, we generated and validated 500 tasks. For each task, we aim to generate 256 trajectories. After aggregating and filtering the trajectories we obtain around 34000 trajectories, which forms our SFT training set. Figure 3 shows a sample of the tasks generated as well as the count of valid paths generated from the tasks. Figure 4 summarize the trajectories in the final training dataset.

Model training We train two models, Qwen3-4B and Qwen3-8B using SFT on the filtered trajectories. Detailed training hyperparameters are available in appendix.

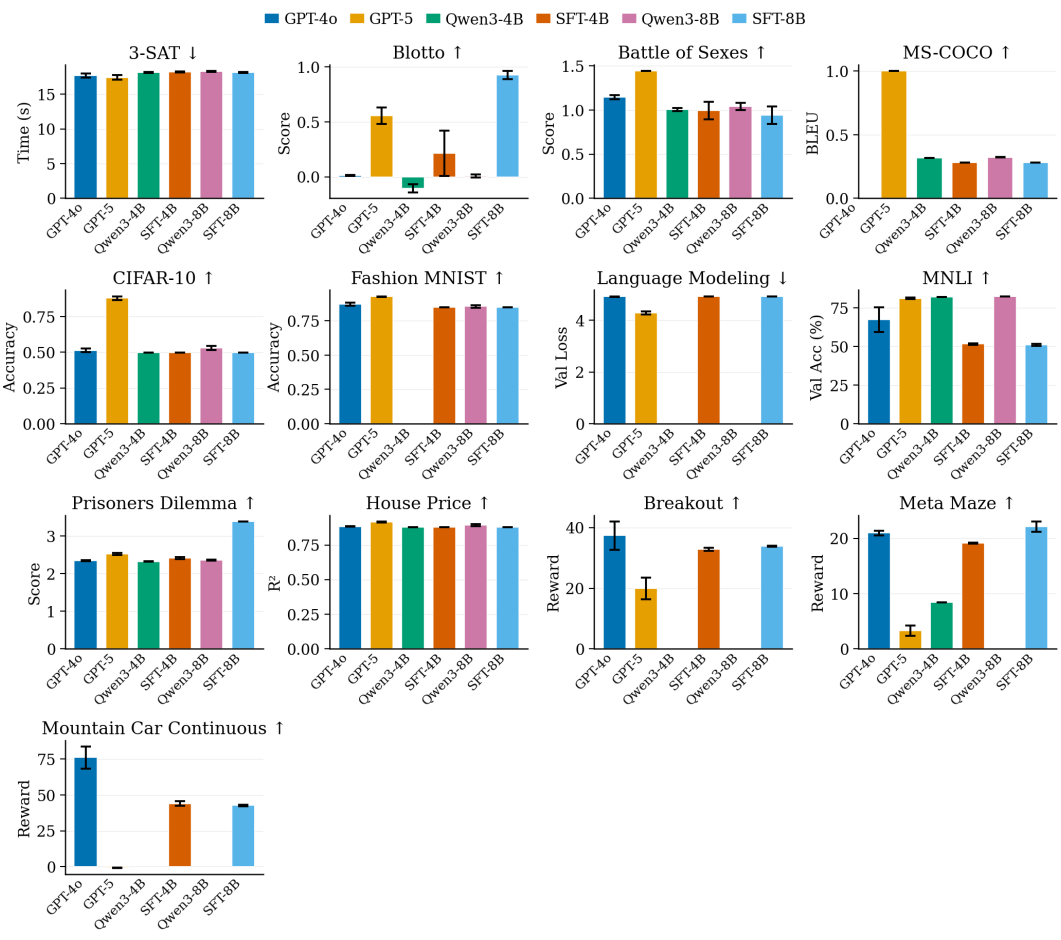

Figure 5: Model performance comparison between the baselines: GPT-4o, GPT-5, Qwen3-4B and Qwen3-8B, and our trained models: SFT-Qwen3-4B and SFT-Qwen8B. The performance is aggregated across 64 runs, which is displayed as violin plots for each subtask of MLGym. If all of the tasks fail, then the chart would show empty bar. In 9 out of 13 tasks, our trained models perform better than the baseline Qwen3-4B models.

We measure the performance of the trained models on the MLGym benchmark, and compare with GPT-4o (OpenAI et al., 2024), GPT-5 (Singh et al., 2024), Qwen3-4B and Qwen3-8B (Yang et al., 2025a). We report the performance on individual tasks and in aggregate in Figure 5 and 6.

4 Discussion

Failure modes Our current task synthesis pipeline covers most but not all tasks in MLGym. For example, for the MS-COCO task, we don't see a performance increase. This is likely because our task synthesis pipeline does not cover well the distribution of more complex starter code files. One direction is to condition the task synthesis on existing, high quality code bases (e.g. NanoGPT), so we can generate more complex tasks.

Extending to other benchmarks Our task synthesis pipeline is fully generic and can be easily extended to other agentic coding tasks. One good fit is MLE-Bench Chan et al. (2025), which uses Kaggle challenges. Since our models are trained on a wide variety of machine learning tasks, we expect to zero-shot performance gains on MLE-Bench.

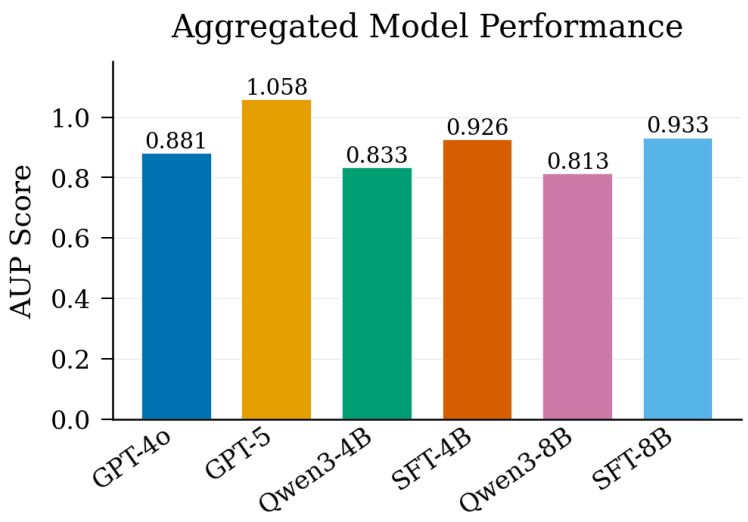

Figure 6: The aggregate performance on MLGym. Since different sub-tasks in MLGym have different score scale and comparison direction, Nathani et al. (2025) introduced the AUP score, which stands for area under the performance curve. Here we report the AUP score of each of the models.

Optimizing for discovery of new ideas While our synthetic task pipeline is a first step towards training LLM agents capable of machine learning tasks, we can explicitly encourage agents to form new ideas during the trajectory sampling by enabling literature search over existing machine learning research.

Reinforcement learning Although all of our model training is done with SFT, our synthetic tasks also can be used for reinforcement learning, where the reward signal is directly the final score defined by the task. Applying RL to machine learning tasks is challenging, because each roll-out may include long GPU training jobs, and the final reward may have vastly different scales. Addressing these challenges is a promising future direction.

5 Related Work

Recent work has explored using LLM-based agents to support scientific research across ideation, execution, and evaluation. For ideation, multi-agent systems such as AI Co-Scientist generate and iteratively refine hypotheses aligned to researcher goals Gottweis et al. (2025). Controlled comparisons suggest LLMs can produce ideas judged more novel than expert proposals, but often with reduced feasibility Siegel et al. (2024), and downstream studies find a pronounced ideation–execution gap when researchers attempt to implement LLM-generated ideas Si et al. (2025). Other efforts structure hypothesis generation explicitly, e.g., via Bit–Flip supervision that links assumptions to counterproposals O'Neill et al. (2025).

To evaluate execution capabilities, several benchmarks test whether agents can reproduce real ML engineering and research workflows. MLE-Bench samples Kaggle-style end-to-end engineering tasks Chan et al. (2025), while PaperBench measures replication of modern ICML papers via many rubric-graded subtasks Starace et al. (2025). Related benchmarks probe targeted execution skills, such as re-implementing and improving training-script optimizations in NanoGPT "speedruns" Zhao et al. (2025). For software engineering, SWE-Smith scales task generation by synthesizing test-breaking instances across Python codebases and improves performance on SWE-bench Verified Yang et al. (2025b).

Finally, work on automated reviewing and end-to-end pipelines highlights both promise and limitations. DeepReview trains reviewer-style models with structured retrieval and argumentation Zhu et al. (2025), whereas broader evaluations show LLM reviewers remain imperfect, especially on long-context understanding and critical feedback Zhou et al. (2024). Toward full research automation, The

AI Scientist-v2 demonstrates hypothesis-to-paper loops with automated experimentation and writing Lu et al. (2024). Benchmarks such as MLAgentBench, MLGym/MLGym-Bench, and MLRC-Bench further study long-horizon research behaviors, generally finding that agents can tune and execute established pipelines but still struggle with robust planning and genuinely novel method discovery Huang et al. (2024); Nathani et al. (2025); Zhang et al. (2025); Chen et al. (2025).

6 Conclusion

We presented a scalable pipeline for training machine learning research agents via *synthetic task scaling*. Our approach automatically generates diverse ML tasks compatible with the SWE-agent framework by sampling topics, proposing and validating real HuggingFace datasets, and synthesizing full runnable environments including configs, starter code, and evaluation scripts. To ensure task validity at scale, we introduced an automated verification and self-debugging loop that filters out broken environments without requiring human intervention.

Using this pipeline, we generated roughly 500 synthetic ML tasks and collected $\sim$30k–34k teacher trajectories from GPT-5. Fine-tuning Qwen3-4B and Qwen3-8B on these trajectories leads to consistent gains on the MLGym benchmark, improving aggregate AUP by 9% and 12% respectively, and improving performance on the majority of individual tasks. These results suggest that synthetic environments can provide effective training signal for long-horizon agent behaviors such as iterative debugging, experimentation, and implementation refinement.

More broadly, our work supports a practical direction for building AI scientists: instead of relying purely on static corpora of papers and code, we can train agents through large-scale experience in executable research environments. We hope this enables future work on reinforcement learning over ML tasks, richer task distributions grounded in real-world codebases, and agents that move beyond optimization toward genuine discovery.

References

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. URL https://arxiv.org/abs/2410.07095.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. Mlr-bench: Evaluating ai agents on open-ended machine learning research. *arXiv preprint arXiv:2505.19955*, 2025. 201 tasks over CS; end-to-end research pipeline; idea+writing ok, experiments often fabricated.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, Yossi Matias, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R D Costa, José R Penadés, Gary Peltz, Yunhan Xu, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Towards an ai co-scientist, 2025. URL https://arxiv.org/abs/2502.18864.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation, 2024. URL https://arxiv.org/abs/2310.03302.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. Mlgym: A new framework and benchmark for advancing ai research agents, 2025. URL https://arxiv.org/abs/2502.14499.

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

Charles O'Neill, Tirthankar Ghosal, Roberta Răileanu, Mike Walmsley, Thang Bui, Kevin Schawinski, and Ioana Ciucă. Sparks of science: Hypothesis generation using structured paper data, 2025. URL https://arxiv.org/abs/2504.12976.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick

Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Chenglei Si, Tatsunori Hashimoto, and Diyi Yang. The ideation-execution gap: Execution outcomes of llm-generated versus human research ideas, 2025. URL https://arxiv.org/abs/2506.20803.

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

Aaditya Singh, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. Openai gpt-5 system card, 2024. URL https://arxiv.org/abs/2410.21276.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, Tejal Patwardhan, and OpenAI. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025. 20 ICML Spotlight/Oral papers; 8,316 sub-tasks; agent score 21%.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Leret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software engineering agents, 2025b. URL https://arxiv.org/abs/2504.21798.

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

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

Ruiyang Zhou, Lu Chen, and Kai Yu. Is LLM a reliable reviewer? a comprehensive evaluation of LLM on automatic paper reviewing tasks. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 9340–9351, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.816/.

Minjun Zhu, Yixuan Weng, Linyi Yang, and Yue Zhang. Deepreview: Improving llm-based paper review with human-like deep thinking process, 2025. URL https://arxiv.org/abs/2503.08569.

A APPENDIX

A.1 PROMPTS USED IN THE TASK GENERATION PIPELINE

This appendix lists the core, non-redundant prompt texts used in the data generation pipeline.

A.1.1 TOPIC SAMPLING PROMPT

```
You are an expert in machine learning research. Generate a list of
    20 diverse and interesting machine learning research topics.
    Each topic should be a short phrase or sentence, suitable for
    use as a research challenge or task. Do not repeat topics from
    previous examples. Return the topics as a JSON array of strings.

Your output:
```

A.2 TASK PROPOSAL AND DATASET VALIDATION PROMPTS

Task proposal prompts

```
You are an expert ML researcher create a training task for a junior
    researcher. Given a topic, generate a JSON object describing a
    machine learning task for that topic. The JSON must include:
- topic: the original topic
- metric: a suitable evaluation metric for the task
- description: a detailed description of the ML task
- dataset: a dataset name that can be matched to a huggingface
    dataset OR a simple search query for a public huggingface
    dataset (e.g., 'cifar10', 'imdb', 'tiny imagenet'). Omit this
    field if the task does not require a dataset.

You have access to a tool that can search the HuggingFace datasets
    API to find suitable datasets based on your query. Please make
    sure the dataset exists on HuggingFace.

Here are some examples:

Topic: Image Classification
Output:
{"topic": "Image Classification", "metric": "Accuracy", "description
    ": "Classify images into categories using a standard image
    classification dataset.", "dataset": "cifar10"}

Topic: Sentiment Analysis
Output:
{"topic": "Sentiment Analysis", "metric": "Accuracy", "description":
     "Predict the sentiment (positive/negative) of movie reviews.",
    "dataset": "imdb"}

Topic: Named Entity Recognition
Output:
{"topic": "Named Entity Recognition", "metric": "F1-score", "
    description": "Identify named entities in text using a standard
    NER dataset.", "dataset": "conll2003"}

Topic: Text Summarization
Output:
{"topic": "Text Summarization", "metric": "ROUGE score", "
    description": "Generate concise summaries of news articles.", "
    dataset": "cnn_dailymail"}
```

```
Topic: Machine Translation
Output:
{"topic": "Machine Translation", "metric": "BLEU", "description": "
    Translate sentences from English to German.", "dataset": "wmt14
    "}

Topic: Speech Command Recognition
Output:
{"topic": "Speech Command Recognition", "metric": "Accuracy", "
    description": "Classify spoken commands from audio clips.", "
    dataset": "google speech commands"}

Topic: Human Activity Recognition
Output:
{"topic": "Human Activity Recognition", "metric": "Accuracy", "
    description": "Classify human activities from wearable sensor
    data.", "dataset": "UCI HAR"}

Topic: {{topic}}
Output:
```

Dataset validation prompt

```
You may call the dataset search tool to validate or refine the
    dataset choice before producing the final JSON.
Rules:
- If you are unsure about the dataset name, call the tool with a
    short query.
- When confident, output ONLY the final JSON object (no surrounding
    prose) with required keys.
- Keys: topic, metric, description, optional dataset (string). If
    dataset provided, ensure it plausibly exists on HuggingFace.
- Avoid re-calling the tool if current results already contain a
    suitable dataset.
- You can modify the topic slightly to fit the available datasets.
```

Dataset Search Tool-Result Follow-Up Prompt

```
Search results for query '{query}': {json.dumps(results,
    ensure_ascii=False)}
Select one dataset id (or refine by calling the tool again) and
    output final JSON when ready.
```

JSON-Missing Nudge Prompt

```
I did not receive a valid JSON. Please either call the search tool
    or output the final JSON object now.
```

A.3 TASK FILES GENERATION PROMPT

Task files stage 1: config generation

```
Your objective is to create YAML config files for a machine learning
     task. You are given JSON input that describes the topic as well
```

```
     as the dataset you are working with. The first file is a task
    configuration file that describes the task, dataset, and
    submission format. The other files (usually one but can be
    multiple) are dataset configuration files that describe the
    datasets used in the task. Be creative and generate a task that
    is interesting and challenging for the agent to solve.

IMPORTANT OUTPUT FORMAT REQUIREMENT:
Return every file using markdown code blocks ONLY (no sentinel
    markers, no extra text) exactly like:
```lang
# relative/path/to/file
<file contents>
```

- The task is executed in a linux environment with Python 3.10 and
    the following packages preinstalled (generic_conda_requirements.
    txt):
 generic_conda_requirements.txt (preinstalled packages):
  numpy
  pandas
  scipy
  torch
  scikit-learn
  tqdm
  datasets
  gymnasium
  transformers[torch]
  datasets
  matplotlib
  torchvision

Here is the format for the input JSON.
```json
# input.json
{
  "topic": ...,
  "metric": ...,
  "description": ...,
  "dataset": {
    "id": ...,
    "features": [...],
    "examples": [...]
  }
}
```

Here is the format for the config files (showing expected file
    outputs using markdown code blocks):
```yaml
# tasks/task_id.yaml
id: # task id, will be used as the config file name as well.
    Recommend using snake_case.
name: # task name
description: # Description of the task that includes the task
    objective submission format requirements. You must include the
    string "{dataset_docs}", which reference the description in the
    dataset config file.
dataset_configs: # zero or more data config files described below.
task_entrypoint: # one of four values: CSVSubmissionTasks,
    ModelSubmissionTasks, LMSubmissionTasks, PythonSubmissionTasks
```
```

````
training_timeout: # timeout in seconds, make a good effort
    estimating the time it takes to train the model on our hardware
    (NVIDIA RTX A6000 GPU, 8 CPU cores).
use_generic_conda: # true if the task does not require any packages
    other than the ones listed in generic_conda_requirements.txt,
    false otherwise
requirements_path: # path to requirements.txt if use_generic_conda
    is false, otherwise leave this empty
starter_code: [] # Leave this as an empty list, it will be filled in
     later
baseline_paths: [] # Leave this empty, it will be filled in later
baseline_scores: [] # Leave this as an empty list, it will be filled
     in later
evaluation_paths: [] # Leave this empty, it will be filled in later
evaluation_read_only: # Whether the evaluation script should be read
    -only to the agent
memory_path: memory.json # This value is fixed
```

```yaml
# datasets/dataset_name.yaml
data_path: # Path to the dataset, IMPORTANT: This must be a valid
    public huggingface dataset, for example "uoft-cs/cifar10" or "
    ILSVRC/imagenet-1k". You can find the dataset ID in the provided
     JSON input. Do not use a placeholder.
description: # detailed description of the dataset, including
    features, content, format, number of classes and samples, and
    any other relevant information. Give concrete example rows of
    the dataset.
is_local: # should always be false
name: # dataset name
```

You can only use one of four values for `task_entrypoint` outlined
    below.

## Quick Decision Flow

* Agent outputs a CSV predictions file?  CSVSubmissionTasks
* Agent submits a model/checkpoint + YAML config?
    ModelSubmissionTasks
* Language-model training/eval that should run with torchrun on GPUs
    ?  LMSubmissionTasks
* Deliverable is Python code you evaluate directly?
    PythonSubmissionTasks

Set this with task_entrypoint in your TaskConfig.

---
1) CSVSubmissionTasks
Use when: The submission is a CSV of predictions (Kaggle-style).
Submission expected: submission.csv in the task workspace root.
Evaluation call (first path in evaluation_paths):
 python <eval_script> --submission_file <path/to/submission.csv>
Eval output format: Entire stdout must be a valid JSON object.
Baseline: If baseline_paths is set, runs the first baseline script,
    then evaluate().
Config snippet:
 task_entrypoint: CSVSubmissionTasks

2) ModelSubmissionTasks
Use when: The agent submits a model artifact or config (e.g.,
    checkpoints + a YAML config), not a CSV.
````
````

```
Submission expected: The first *.yaml file found under the workspace
    .
Evaluation call:
 python <eval_script> --config_fname <path/to/config.yaml>
Eval output format: Stdout must contain at least one line starting
    with { that is valid JSON.
Baseline: Same approach.
Config snippet:
 task_entrypoint: ModelSubmissionTasks

3) LMSubmissionTasks
Use when: Language-model tasks that should run distributed via
    torchrun.
Evaluation call:
 torchrun --nproc_per_node=<detected_gpus> --standalone <eval_script
     >
Eval output format: First line starting with { that parses as JSON.
Baseline: Also with torchrun.
Config snippet:
 task_entrypoint: LMSubmissionTasks

4) PythonSubmissionTasks
Use when: The agent writes Python code that evaluator imports/
    executes directly.
Submission expected: target.py file.
Evaluation call:
 python <eval_script>
Eval output format: Entire stdout JSON object.
Config snippet:
 task_entrypoint: PythonSubmissionTasks
---

Important tips:
1. Make the task and dataset description as detailed as possible:
    task objective, dataset format, data examples, submission format
    , metrics, constraints. Be very informative because the agent
    rely on this information.
2. Include full examples of data rows in the dataset config
    description.
3. The "id" field of the task config must be exactly same as the
    filename without the .yaml extension.
4. In the description, always escape curly braces with double braces
     {{ and }}, except for {dataset_docs}.
5. You may see a dataset field in the input JSON, use that as the
    data_path in the dataset config. Use the dataset ID from the
    input JSON exactly as the dataset name, it is a verified public
    huggingface dataset.
6. Use the dataset information provided to you (if any), give
    detailed information about the features, content, and example
    rows of the dataset (if any).
7. If you are not prompted with a dataset, optionally choose a valid
     public huggingface dataset. DO NOT emit a placeholder or other
    invalid names of the dataset.
8. Follow the task description provided in the input.
9. Output files in order: one task config, then dataset config files
    . Only one task config file.

Example description:
{{example_input_1}}

Example output:
{{example_1}}
```

```
Example description:
{{example_input_2}}

Example output:
{{example_2}}

Your description:
{{task_description}}

Think step by step and plan out the task before writing the output
    files.
Your output:
```

Task files stage 2: starter code generation

```
You are tasked to create a ML training task for a autonomous machine
     learning research agent according to given config files.
The agent is an autonomous Machine Learning Researcher operating in
    a specialized command-line environment.
In a turn-based interaction, the agent provides a "discussion" of
    its plan, followed by a single shell command.
The agent can navigate the file system, and read and write files
    using special commands, but must handle code indentation
    manually.
The agent is provided with baseline code for an ML task and its goal
     is to improve the model's performance and submit the final
    solution.

The input task config file look like this:
```yaml
# tasks/task_id.yaml
id: # task id, will be used as the config file name as well.
    Recommend using snake_case.
name: # task name
description: # Description of the task that includes the task
    objective submission format requirements. You must include the
    string "{dataset_docs}", which reference the description in the
    dataset config file.
dataset_configs: # zero or more data config files described below.
task_entrypoint: # one of four values: CSVSubmissionTasks,
    ModelSubmissionTasks, LMSubmissionTasks, PythonSubmissionTasks
training_timeout: # timeout in seconds, make a good effort
    estimating the time it takes to train the model on our hardware
    (NVIDIA RTX A6000 GPU, 8 CPU cores).
use_generic_conda: # true if the task does not require any packages
    other than the ones listed in generic_conda_requirements.txt,
    false otherwise
requirements_path: # path to requirements.txt if use_generic_conda
    is false, otherwise leave this empty
starter_code: [] # Fill this in after creating the task files
baseline_paths: [] # You need to identify the baseline file and fill
     this in
baseline_scores: [] # Lease this empty, it will be filled in later
    by running your evaluation
evaluation_paths: [] # You need to identify the evaluation file and
    fill this in
evaluation_read_only: # Whether the evaluation script should be read
    -only to the agent
memory_path: memory.json # This value is fixed
```
```

````
The input dataset configs are optional, and look like this:
```yaml
# datasets/dataset_name.yaml
data_path: # Path to the dataset, IMPORTANT: This must be a valid
    public huggingface dataset, for example "uoft-cs/cifar10" or "
    ILSVRC/imagenet-1k". Do not use a placeholder.
description: # detailed description of the dataset, including
    content, format, number of classes and samples, and any other
    relevant information
is_local: # should always be false
name: # dataset name
```

IMPORTANT OUTPUT FORMAT REQUIREMENT:
Return every file using markdown code blocks ONLY exactly like:
```lang
# relative/path/to/file
<file contents>
```

Output ALL created files this way.

The task is executed in a linux environment with Python 3.10 and the
     following packages preinstalled (generic_conda_requirements.txt
    ):
 generic_conda_requirements.txt (preinstalled packages):
  numpy
  pandas
  scipy
  torch
  scikit-learn
  tqdm
  datasets
  gymnasium
  transformers[torch]
  datasets
  matplotlib
  torchvision
If you decides to use any other packages, you MUST include a file
    named requirements.txt in the output, and set the
    use_generic_conda field to false in the task config.


Follow the following instructions:
- The input gives you a task config + dataset config(s). You must
    produce runnable starter code.
- Leave the starter_code field empty, I will help you fill it.
- Provide exactly one baseline file path in baseline_paths; baseline
     must be directly runnable without any command line arguments
    and generate a valid submission artifact.
- Provide exactly one evaluation file path in evaluation_paths;
    IMPORTANT: evaluation must output a JSON with a single field and
     numeric score to stdout.
- Do not fill baseline_scores, it will be filled in later by running
     the baseline and evaluation.
- You MAY create auxiliary scripts (data utils, model, etc.)
- Evaluation file MUST print a single valid JSON object with string
    keys and float values (only once) for metrics.
- Task must run in under 30 minutes on a NVIDIA RTX A6000 GPU and 8
    CPU cores.
- If you set use_generic_conda: true, then use only the preinstalled
     packages. If you set use_generic_conda: false you add
````

```
    requirements.txt, the path to requirements.txt MUST be set in
    the task config.
- Choose dataset usage consistent with provided dataset config(s)
    and task type. Remember you MUST use a REAL and VALID public
    huggingface dataset. Your task may not need a dataset (e.g. game
     theory).
- Keep the baseline simple, leave room for the agent to improve it.
- If the user runs into errors validating the task, you can change
    the task config to fix the issue.


When you create the evaluation script, if will be ran with different
     commands based on the value of the task_entrpoint field. The
    output must print a valid JSON object. Your evaluation file must
     respect the evaluation call format for the task class.
---
1) CSVSubmissionTasks
Use when: The submission is a CSV of predictions (Kaggle-style).
Submission expected: submission.csv in the task workspace root.
Evaluation call (first path in evaluation_paths):
 python <eval_script> --submission_file <path/to/submission.csv>

2) ModelSubmissionTasks
Use when: The agent submits a model artifact or config (e.g.,
    checkpoints + a YAML config), not a CSV.
Submission expected: The first *.yaml file found under the workspace
    .
Evaluation call:
 python <eval_script> --config_fname <path/to/config.yaml>

3) LMSubmissionTasks
Use when: Language-model tasks that should run distributed via
    torchrun.
Evaluation call:
 torchrun --nproc_per_node=<detected_gpus> --standalone <eval_script
     >
Eval output format: First line starting with { that parses as JSON.

4) PythonSubmissionTasks
Use when: The agent writes Python code that evaluator imports/
    executes directly.
Submission expected: target.py file.
Evaluation call:
 python <eval_script>
---

Coding tips:
- Use multiprocessing / DataLoader workers for speed. You have 8 CPU
     cores.
- Use GPUs for training and evaluation and anything else that makes
    sense, assume it is always available. You have a NVIDIA RTX
    A6000 GPU.
- Use deterministic seeds where relevant.
- Use the right indentation.
- Import the dataset using the datasets library load_dataset
    function.
- Do not import scripts that you have not written.
- Do not import packages that are not in the
    generic_conda_requirements.txt or requirements.txt.
- For your ease, strongly prefer a flat directory structure, and
    create subfolders only if necessary.
- Use the dataset_path variable in the dataset config given to you.
    It is verified to be a valid public huggingface dataset.
```

```
- Try to use as flexible as possible package requirements, e.g. "
    torch>=2.0.0" instead of "torch==2.0.0" in requirements.txt,
    since you may have outdated knowledge.

Here is an example:
{{example_input_1}}
Example output:
{{example_output_1}}

Here is another example:
{{example_input_2}}
Example output:
{{example_output_2}}

Here is your task config:
{{task_config}}

Think step by step and plan out the task before writing the code
    files.
Your output:
```

Error-Recovery Retry Prompt

```
Error encountered: {self.stage_1_err}

Please try again. Return the revised output in whole.
```

A.4 EXAMPLE SYNTHETIC TASK

We show a random example among our generated tasks. The task includes

1. Task description `hotpotqa_join_facts_qa.yaml`
2. Dataset description `hotpotqa_hotpot_qa.yaml`
3. Starting implementation `baseline.py`
4. Evaluation code `evaluate.py`

hotpotqa_hotpot_qa.yaml hotpotqa_join_facts_qa.yaml

```
data_path: hotpotqa/hotpot_qa
description: "HotpotQA is a large-scale multi-hop question answering
     dataset featuring\
  \ questions that require reasoning across multiple documents. This
       configuration\
  \ targets the distractor setting, where each example provides 10
      candidate paragraphs\
  \ (titles and sentence lists), of which only a subset contains the
       gold supporting\
  \ sentences needed to answer the question.\nFeatures: - id (string
     ): Unique identifier\
  \ for the example, e.g., \"5a7a06935542990198eaf050\". - question
     (string): Natural\
  \ language question requiring multi-hop reasoning. - answer (
     string): Gold answer\
  \ text (can be \"yes\"/\"no\" or a short span). - type (string):
     Question type,\
```

```
\ e.g., \"comparison\", \"bridge\". - level (string): Difficulty
    level, e.g., \"\
easy\", \"medium\", \"hard\". - supporting_facts (struct of lists)
    :\n - title\
\ (list[string]): Titles of the documents containing supporting
    sentences.\n \
\ - sent_id (list[int32]): 0-based indices of the supporting
    sentences in the corresponding\
\ documents.\n The k-th title aligns with the k-th sent_id to form
     a pair (title[k],\
\ sent_id[k]).\n- context (struct):\n - title (list[string]):
    Titles of the 10\
\ candidate documents.\n - sentences (list[list[string]]): For
    each document,\
\ a list of its sentence strings, aligned by index with context.
    title.\n\nTypical\
\ splits: - train: ~90k-113k examples (depending on release/
    version). - validation/dev:\
\ ~7k-8k examples. - test: may be available without supporting
    facts/answers in\
\ certain releases. For this task, use train and validation/dev.\
    nData format details:\
\ - Sentence indices in supporting_facts.sent_id are 0-based and
    reference the sentence\
\ array of the document whose title matches supporting_facts.title.
     - The distractor\
\ setting includes 10 documents (context.title length == 10); each
     has a variable\
\ number of sentences. - Answer normalization for EM/F1 is
    performed during evaluation\
\ (lowercasing, removing punctuation and articles).\nConcrete
    examples: - Example\
\ 1:\n id: \"5a7a06935542990198eaf050\"\n question: \"Which
    magazine was started\
\ first Arthur's Magazine or First for Women?\"\n answer: \"Arthur'
    s Magazine\"\
\n type: \"comparison\"\n level: \"medium\"\n supporting_facts:\n
    title: [\"\
Arthur's Magazine\", \"First for Women\"]\n sent_id: [0, 0]\n
    context:\n \
\ title: [\n \"Radio City (Indian radio station)\", \"History of
    Albanian football\"\
, \"Echosmith\",\n \"Women's colleges in the Southern United
    States\", \"First\
\ Arthur County Courthouse and Jail\",\n \"Arthur's Magazine\",
    \"2014-15 Ukrainian\
\ Hockey Championship\", \"First for Women\",\n \"Freeway Complex
    Fire\", \"\
William Rast\"\n ]\n sentences: [\n [\n \"Radio City is India's\
\ first private FM radio station and was started on 3 July
    2001.\",\n \"\
It broadcasts on 91.1 (earlier 91.0 in most cities) megahertz from
     Mumbai (where\
\ it was started in 2004), Bengaluru (started first in 2001),
    Lucknow and New Delhi\
\ (since 2003).\",\n \"It plays Hindi, English and regional songs
    .\",\n \
\ \"It was launched in Hyderabad in March 2006, in Chennai on 7
    July 2006\
\ and in Visakhapatnam October 2007.\",\n \"Radio City recently
    forayed into\
\ New Media in May 2008 with the launch of a music portal -
    PlanetRadiocity.com\
```

```
\ that offers music related news, videos, songs, and other music-
    related features.\"\
,\n \"The Radio station currently plays a mix of Hindi and
    Regional music.\"\
,\n \"Abraham Thomas is the CEO of the company.\"\n ],\n ...\n\
\ [\n \"Arthur's Magazine (1844-1846) was an American literary
    periodical\
\ published in Philadelphia in the 19th century.\",\n \"Edited by
    T.S. Arthur,\
\ it featured work by Edgar A. Poe, J.H. Ingraham, Sarah Josepha
    Hale, Thomas G.\
\ Spear, and others.\",\n \"In May 1846 it was merged into \\\"
    Godey's Lady's\
\ Book\\\".\"\n ],\n ...\n [\n \"First for Women is a woman's\
\ magazine published by Bauer Media Group in the USA.\",\n \"The
    magazine\
\ was started in 1989.\",\n \"It is based in Englewood Cliffs, New
     Jersey.\"\
,\n \"In 2011 the circulation of the magazine was 1,310,696 copies
    .\"\n \
\ ],\n ...\n ]\n\n- Example 2:\n id: \"5a879ab05542996e4f30887e\"\
    n\
\ question: \"The Oberoi family is part of a hotel company that
    has a head office\
\ in what city?\"\n answer: \"Delhi\"\n type: \"bridge\"\n level:
    \"medium\"\n\
\ supporting_facts:\n title: [\"Oberoi family\", \"The Oberoi
    Group\"]\n \
\ sent_id: [0, 0]\n context:\n title: [\n \"Ritz-Carlton Jakarta\",
     \"\
Oberoi family\", \"Ishqbaaaz\", \"Hotel Tallcorn\", \"Mohan Singh
    Oberoi\",\n \
\ \"Hotel Bond\", \"The Oberoi Group\", \"Future Fibre
    Technologies\", \"289th\
\ Military Police Company\",\n \"Glennwanis Hotel\"\n ]\n
    sentences:\
\ [\n [\n \"The Ritz-Carlton Jakarta is a hotel and skyscraper in
    Jakarta,\
\ Indonesia and 14th Tallest building in Jakarta.\",\n \"It is
    located in\
\ city center of Jakarta, near Mega Kuningan, adjacent to the
    sister JW Marriott\
\ Hotel.\",\n \"It is operated by The Ritz-Carlton Hotel Company
    .\",\n \
\ \"The complex has two towers that comprises a hotel and the
    Airlangga Apartment\
\ respectively.\",\n \"The hotel was opened in 2005.\"\n ],\n [\n\
\ \"The Oberoi family is an Indian family that is famous for its
    involvement\
\ in hotels, namely through The Oberoi Group.\"\n ],\n ...,\n [\n\
\ \"The Oberoi Group is a hotel company with its head office in
    Delhi.\"\
,\n \"Founded in 1934, the company owns and/or operates 30+ luxury
     hotels\
\ and two river cruise ships in six countries, primarily under its
     Oberoi Hotels\
\ & Resorts and Trident Hotels brands.\"\n ],\n ...\n ]\n\n-
    Example\
\ 3:\n id: \"5a8d7341554299441c6b9fe5\"\n question: \"Musician and
     satirist Allie\
\ Goertz wrote a song about the \\\"The Simpsons\\\" character
    Milhouse, who Matt\
```

```
\ Groening named after who?\"\n answer: \"President Richard Nixon
    \"\n type: \"\
bridge\"\n level: \"hard\"\n supporting_facts:\n title: [\"Allie
    Goertz\",\
\ \"Allie Goertz\", \"Allie Goertz\", \"Milhouse Van Houten\"]\n
    sent_id: [0,\
\ 1, 2, 0]\n context:\n title: [\n \"Lisa Simpson\", \"Marge
    Simpson\"\
, \"Bart Simpson\", \"Allie Goertz\", \"Milhouse Van Houten\",\n
    \"Los Angeles\
\ Reader\", \"Homer Simpson\", \"List of The Simpsons video games
    \",\n \"The\
\ Simpsons: An Uncensored, Unauthorized History\", \"List of The
    Simpsons guest\
\ stars\"\n ]\n sentences: [\n [\n \"Lisa Marie Simpson is a
    fictional\
\ character in the animated television series \\\"The Simpsons
    \\\".\",\n \
\ \"She is the middle child and most intelligent of the Simpson
    family.\",\n \
\ \"Voiced by Yeardley Smith, Lisa first appeared on television in
     \\\"The Tracey\
\ Ullman Show\\\" short \\\"Good Night\\\" on April 19, 1987.\",\n
     \"Cartoonist\
\ Matt Groening created and designed her while waiting to meet
    James L. Brooks.\"\
,\n \"Groening had been invited to pitch a series of shorts based
    on his\
\ comic \\\"Life in Hell\\\", but instead decided to create a new
    set of characters.\"\
,\n \"He named the elder Simpson daughter after his younger sister
     Lisa Groening.\"\
,\n \"After appearing on \\\"The Tracey Ullman Show\\\" for three
    years,\
\ the Simpson family were moved to their own series on Fox, which
    debuted on December\
\ 17, 1989.\"\n ],\n ...,\n [\n \"Allison Beth \\\"Allie\\\
\" Goertz (born March 2, 1991) is an American musician.\",\n \"
    Goertz is\
\ known for her satirical songs based on various pop culture
    topics.\",\n \
\ \"Her videos are posted on YouTube under the name of
    Cossbysweater.\",\n \
\ \"Subjects of her songs have included the film \\\"The Room\\\",
     the character\
\ Milhouse from the television show \\\"The Simpsons\\\", and the
    game Dungeons\
\ & Dragons.\",\n \"Her style has been compared to that of Bo
    Burnham.\"\
,\n \"In December 2015, Goertz released a concept album based on
    the Adult\
\ Swim series \\\"Rick and Morty\\\", \\\"Sad Dance Songs\\\",
    with the album's\
\ cover emulating the animation and logo of the series.\",\n \"The
     album\
\ was made possible through Kickstarter.\",\n \"She is co-host of
    Everything's\
\ Coming Up Podcast, a Simpsons-focused podcast along with Julia
    Prescott.\"\n \
\ ],\n [\n \"Milhouse Mussolini van Houten is a fictional
    character\
\ featured in the animated television series \\\"The Simpsons\\\",
     voiced by Pamela\
```

```
  \ Hayden, and created by Matt Groening who named the character
     after President Richard\
  \ Nixon's middle name.\",\n \"Later in the series, it is revealed
     that Milhouse's\
  \ middle name is \\\"Mussolini.\\\"\"\n ],\n ...\n ]\n\nNotes: -
     Titles\
  \ must match exactly when referencing supporting facts. - sent_id
     values must be\
  \ valid indices into the corresponding document's sentence list. -
      The dataset is\
  \ English-only and licensed under CC BY-SA 4.0.\n"
is_local: false
name: hotpotqa/hotpot_qa
```

hotpotqa_join_facts_qa.yaml

```
   id: hotpotqa_joint_facts_qa
name: HotpotQA Multi-hop QA with Supporting Facts (Distractor)
description: |
 Build a multi-hop QA system on the HotpotQA distractor setting
     that predicts both the final answer and the exact set of
     supporting sentences used to derive it. Each example provides
     a question, 10 candidate paragraphs (title, list of sentences),
      gold supporting facts as {{title, sent_id}} pairs, and an
     answer string {{which may be yes/no or a short span}}. You
     must use the provided sentence boundaries exactly {{do not re-
     split}} and select sentence indices from the given context.
     Train a multi-task model that: {{1}} predicts the answer, and
     {{2}} performs sentence-level classification over all
     candidate sentences to select supporting facts. Use either an
     extractive span head {{start/end indices with a separate yes/
     no head}} or a generative seq2seq model for the answer; use a
     classifier over sentence representations for supporting facts.
      Optimize a weighted sum of answer loss and supporting fact
     classification loss. Evaluate using the official HotpotQA
     metrics: Answer EM/F1, Supporting Facts EM/F1, and Joint EM/F1.
      Report Joint F1 as the primary metric, and include component
     metrics for analysis.
 Data and splits: - Use the train split for training and the
     validation/dev split for model selection and reporting. - This
      task uses the distractor setting {{10 provided paragraphs per
      question, where only a subset contains the supporting facts}}.
      - See {dataset_docs} for full dataset schema, features, and
     concrete examples.
 Modeling guidance: - Represent the 10-paragraph context with
     hierarchical encoders {{e.g., encode sentences, aggregate to
     paragraphs, then across paragraphs}}. - For long inputs,
     consider models that handle long sequences {{e.g., Longformer,
      BigBird}} or multi-hop retrieval to prune the context. -
     Supporting facts prediction is a multi-label classification
     over all candidate sentences in the provided context.
 Submission format: - You must submit a CSV file named submission.
     csv in the workspace root. - Columns:
   - id: the example id string {{e.g., "5a7a06935542990198eaf050"}}.
   - answer: the predicted answer string {{exact text; normalization
       is applied during evaluation}}.
   - supporting_facts: a JSON array of objects, each with keys "
      title" and "sent_id" indicating the selected supporting
      sentences.
 - Example lines:
   - id,answer,supporting_facts
```

```
    - 5a7a06935542990198eaf050,Arthur's Magazine,"[{{""title"": ""
        Arthur's Magazine"", ""sent_id"": 0}}, {{""title"": ""First
        for Women"", ""sent_id"": 0}}]"
    - 5a879ab05542996e4f30887e,Delhi,"[{{""title"": ""Oberoi family
        "", ""sent_id"": 0}}, {{""title"": ""The Oberoi Group"", ""
        sent_id"": 0}}]"
  - Constraints:
    - Each supporting fact must reference a title that exists in the
        example's context.title list and a valid sentence index {{0-
        based}} for that title.
    - No duplicates; order does not matter. Predict exactly the set
        of gold supporting sentences to achieve Supporting Facts EM.
    - Include as many sentences as required by the example {{often 2,
         but some questions require more}}.

  Evaluation and metrics: - Primary metric: Joint F1 {{combines
      answer F1 and supporting facts F1}}. - Secondary metrics
      reported: Answer EM/F1, Supporting Facts EM/F1, Joint EM. -
      Answer normalization follows HotpotQA conventions {{
      lowercasing, stripping punctuation and articles}}. -
      Supporting facts F1 computed over the set of {{title, sent_id}}
       pairs.
  Resources and expected runtime: - Training is expected to complete
       in ~3-5 hours on a single NVIDIA RTX A6000 with 8 CPU cores
      for pre-processing. - Keep memory usage in mind when encoding
      long contexts. Batch size and gradient accumulation may be
      required.
  Tips: - Start with a strong encoder {{e.g., DeBERTa-v3, RoBERTa}}
      with segment-level inputs and a sentence classification head. -
       Use curriculum: begin with answer-only training, then add
      supporting facts loss, or alternate batches. - Joint decoding
      heuristic: prioritize sentences from predicted relevant
      paragraphs; ensure coverage of all hops before final answer
      extraction/generation.
dataset_configs:
- datasets/hotpotqa_joint_facts_qa/hotpotqa_hotpot_qa.yaml
task_entrypoint: CSVSubmissionTasks
training_timeout: 18000
use_generic_conda: true
starter_code:
- data_train_v1/hotpotqa_joint_facts_qa/baseline.py
- data_train_v1/hotpotqa_joint_facts_qa/evaluate.py
baseline_paths:
- baseline.py
baseline_scores:
- joint_f1: 0.022210986997935424
  ans_em: 0.052532072923700206
  ans_f1: 0.08454953694131888
  sp_em: 0.008102633355840648
  sp_f1: 0.12714489351896444
  joint_em: 0.0013504388926401081
evaluation_paths:
- evaluate.py
evaluation_read_only: true
memory_path: data_train_v1/hotpotqa_joint_facts_qa/memory.json
```

baseline.py

```
   import csv
import json
from datasets import load_dataset
```

```
from tqdm import tqdm

YES_NO_STARTS = (
   "is", "are", "was", "were",
   "do", "does", "did",
   "can", "could", "may", "might", "must",
   "have", "has", "had",
   "will", "would", "should", "shall"
)

def simple_answer_heuristic(question, fallback_title):
   q = (question or "").strip().lower()
   if any(q.startswith(aux + " ") for aux in YES_NO_STARTS):
      return "yes"
   return fallback_title if fallback_title is not None else "unknown
       "

def main():
   # Use the distractor setting validation split
   ds = load_dataset("hotpotqa/hotpot_qa", "distractor", split="
       validation")

   with open("submission.csv", "w", newline="", encoding="utf-8") as
        f:
      writer = csv.DictWriter(f, fieldnames=["id", "answer", "
          supporting_facts"])
      writer.writeheader()

      for ex in tqdm(ds, desc="Generating baseline predictions"):
         ex_id = ex["id"]
         question = ex["question"]
         titles = ex["context"]["title"]
         sentences = ex["context"]["sentences"]

         # Choose up to two candidate supporting facts: first two
             titles with at least one sentence
         pred_sfs = []
         for idx, title in enumerate(titles):
            if idx < len(sentences) and len(sentences[idx]) > 0:
               pred_sfs.append({"title": title, "sent_id": 0})
            if len(pred_sfs) == 2:
               break

         # Fallbacks if not enough
         if not pred_sfs:
            # Ensure we still output something structurally valid
            if len(titles) > 0:
               pred_sfs = [{"title": titles[0], "sent_id": 0}]
            else:
               pred_sfs = []

         fallback_title = pred_sfs[0]["title"] if pred_sfs else (
             titles[0] if titles else None)
         answer = simple_answer_heuristic(question, fallback_title)

         writer.writerow({
            "id": ex_id,
            "answer": answer,
            "supporting_facts": json.dumps(pred_sfs, ensure_ascii=
                False)
         })

if __name__ == "__main__":
```

```
    main()
```

evaluate.py

```
import argparse
import csv
import json
import math
import re
import string
from collections import defaultdict
from datasets import load_dataset

PUNCT = set(string.punctuation)
ARTICLES = {"a", "an", "the"}
WHITESPACE_RE = re.compile(r"\s+")

def normalize_answer(s):
   if s is None:
      return ""
   s = s.lower()

   def remove_punc(text):
      return "".join(ch for ch in text if ch not in PUNCT)

   def remove_articles(text):
      return re.sub(r"\b(a|an|the)\b", " ", text)

   def white_space_fix(text):
      return WHITESPACE_RE.sub(" ", text).strip()

   return white_space_fix(remove_articles(remove_punc(s)))

def f1_score(prediction, ground_truth):
   pred_tokens = normalize_answer(prediction).split()
   gold_tokens = normalize_answer(ground_truth).split()
   if len(pred_tokens) == 0 and len(gold_tokens) == 0:
      return 1.0
   if len(pred_tokens) == 0 or len(gold_tokens) == 0:
      return 0.0
   common = defaultdict(int)
   for t in gold_tokens:
      common[t] += 1
   num_same = 0
   for t in pred_tokens:
      if common[t] > 0:
         num_same += 1
         common[t] -= 1
   if num_same == 0:
      return 0.0
   precision = num_same / len(pred_tokens)
   recall = num_same / len(gold_tokens)
   return 2 * precision * recall / (precision + recall)

def exact_match_score(prediction, ground_truth):
   return 1.0 if normalize_answer(prediction) == normalize_answer(
       ground_truth) else 0.0

def sp_em_f1(pred_set, gold_set):
   # pred_set and gold_set are sets of (title, sent_id) tuples
   inter = pred_set.intersection(gold_set)
```

```
    if len(gold_set) == 0 and len(pred_set) == 0:
        return 1.0, 1.0
    if len(gold_set) == 0:
        # No gold facts; treat as EM/F1 zero if pred non-empty;
            otherwise 1
        return (1.0 if len(pred_set) == 0 else 0.0), (1.0 if len(
            pred_set) == 0 else 0.0)
    em = 1.0 if pred_set == gold_set else 0.0
    if len(pred_set) == 0:
        return em, 0.0
    precision = len(inter) / len(pred_set)
    recall = len(inter) / len(gold_set)
    if precision + recall == 0:
        f1 = 0.0
    else:
        f1 = 2 * precision * recall / (precision + recall)
    return em, f1

def parse_supporting_facts(cell):
    try:
        data = json.loads(cell)
        out = set()
        if isinstance(data, list):
            for item in data:
                if isinstance(item, dict):
                    title = item.get("title", "")
                    sent_id = item.get("sent_id", 0)
                    try:
                        sent_id = int(sent_id)
                    except Exception:
                        # if cannot parse, skip this item
                        continue
                    out.add((title, sent_id))
        return out
    except Exception:
        return set()

def load_predictions_csv(path):
    preds = {}
    with open(path, "r", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            ex_id = row.get("id", "")
            answer = row.get("answer", "")
            sf_cell = row.get("supporting_facts", "[]")
            pred_sfs = parse_supporting_facts(sf_cell)
            preds[ex_id] = {"answer": answer, "supporting_facts":
                pred_sfs}
    return preds

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--submission_file", type=str, required=True)
    args = parser.parse_args()

    # Load dev split of HotpotQA distractor
    ds = load_dataset("hotpotqa/hotpot_qa", "distractor", split="
        validation")

    gold_by_id = {}
    for ex in ds:
        ex_id = ex["id"]
        answer = ex["answer"]
```

```
        titles = ex["supporting_facts"]["title"]
        sent_ids = ex["supporting_facts"]["sent_id"]
        gold_sfs = set()
        for t, s in zip(titles, sent_ids):
            try:
                s = int(s)
            except Exception:
                continue
            gold_sfs.add((t, s))
        gold_by_id[ex_id] = {"answer": answer, "supporting_facts":
            gold_sfs}

    preds = load_predictions_csv(args.submission_file)

    total = len(gold_by_id)
    ans_em_sum = 0.0
    ans_f1_sum = 0.0
    sp_em_sum = 0.0
    sp_f1_sum = 0.0
    joint_em_sum = 0.0
    joint_f1_sum = 0.0

    for ex_id, gold in gold_by_id.items():
        pred = preds.get(ex_id, {"answer": "", "supporting_facts": set
            ()})

        a_em = exact_match_score(pred["answer"], gold["answer"])
        a_f1 = f1_score(pred["answer"], gold["answer"])
        s_em, s_f1 = sp_em_f1(pred["supporting_facts"], gold["
            supporting_facts"])

        ans_em_sum += a_em
        ans_f1_sum += a_f1
        sp_em_sum += s_em
        sp_f1_sum += s_f1
        joint_em_sum += (a_em * s_em)
        joint_f1_sum += (a_f1 * s_f1)

    metrics = {
        "joint_f1": joint_f1_sum / total if total > 0 else 0.0,
        "ans_em": ans_em_sum / total if total > 0 else 0.0,
        "ans_f1": ans_f1_sum / total if total > 0 else 0.0,
        "sp_em": sp_em_sum / total if total > 0 else 0.0,
        "sp_f1": sp_f1_sum / total if total > 0 else 0.0,
        "joint_em": joint_em_sum / total if total > 0 else 0.0,
    }
    # Print a single JSON object
    print(json.dumps(metrics))

if __name__ == "__main__":
    main()
```