ML-Dev-Bench: Comparative Analysis of AI Agents on ML development workflows

Harshith Padigela* Chintan Shah* Dinkar Juyal*

1 Abstract

In this report, we present ML-Dev-Bench[1], a benchmark aimed at testing agentic capabilities on applied Machine Learning development tasks. While existing benchmarks focus on isolated coding tasks or Kaggle-style competitions, ML-Dev-Bench tests agents' ability to handle the full complexity of ML development workflows. The benchmark assesses performance across critical aspects including dataset handling, model training, improving existing models, debugging, and API integration with popular ML tools. We evaluate three agents - **ReAct**, **Openhands**, and **AIDE** - on a diverse set of 25 tasks, providing insights into their strengths and limitations in handling practical ML development challenges.

2 Introduction

Recent advances in Large Language Models (LLMs) have demonstrated impressive capabilities in code generation and software engineering tasks. This has led to the development of various benchmarks like HumanEval [2], MBPP [1] that evaluate coding abilities, and others like SWE-Bench [5], that test LLM-based agents on software engineering tasks. However, while these benchmarks effectively assess general programming capabilities, they don't capture the unique challenges of Machine Learning development workflows,

Benchmarks like ML-Bench [6], test agents' abilities to generate code and commands to interact with popular ML repositories, while MLE-Bench [2] and MLAgentBench [4] focus on Kaggle-style tasks to evaluate the iterative and open-ended nature of ML development. However, real-world ML development extends far beyond that, including the complexity of working on top of existing codebases and models, integrating with third-party tools, debugging complex issues that span multiple components of the ML pipeline and understanding and

[0]*Equal contribution

[1]`https://github.com/ml-dev-bench/ml-dev-bench`

Category	Description
Dataset Handling	Downloading and preprocessing datasets
Model Training	Loading pretrained models, fine-tuning
Debugging	Addressing errors in training files, exploding gradients, and incorrect implementations
Model Implementation	Modifying and implementing on top of existing model architectures
API Integration	Integrating logging tools like WandB
Performance	Improving baselines and achieving competitive results

Table 1: Task Categories and Their Descriptions

balancing trade-offs like model performance and cost to come up with optimal design.

ML-Dev-Bench addresses this gap by providing a comprehensive evaluation framework that tests an agent's ability to handle real-world ML development scenarios. Our benchmark is particularly relevant as ML development increasingly relies on large language models and AI agents to assist developers. Understanding the capabilities and limitations of these agents in handling practical ML development tasks is crucial for their effective deployment in production environments.

3 Benchmark Design

ML-Dev-Bench comprises of 25 carefully designed tasks that evaluate various aspects of ML development. These tasks are structured to assess both specific technical capabilities (like handling datasets, model implementation) and broader problem-solving skills (like model training and performance improvement) that are essential in real-world ML development. The tasks span several key categories of ML development shown in Table 1:

1. Dataset Handling focuses on evaluating the ability to work with large datasets, inspect them and apply pre-processing pipelines. An example is the noisy imagenette [3] dataset download task, where the agent needs to download the dataset, inspect its contents to identify the labels file, only load the 50% noisy labels from it and generate class summary statistics.

2. Model Training tests an agent's ability to work with existing models, from loading pretrained weights to implementing training loops, logging metrics

Category	ReAct-Sonnet	OH-Sonnet	Aide-4o	ReAct-4o
Dataset Handling	100% (3/3)	100% (3/3)	33% (1/3)	0% (0/3)
Model Training	67% (4/6)	83% (5/6)	33% (2/6)	50% (3/6)
Debugging	67% (4/6)	67% (4/6)	33% (2/6)	16% (1/6)
API Integration	100% (1/1)	100% (1/1)	0% (0/1)	100% (1/1)
Model Implementation	50% (2/4)	50% (2/4)	0% (0/4)	0% (0/4)
Performance	0% (0/5)	0% (0/5)	0% (0/5)	0% (0/5)
Overall	**56% (14/25)**	**60% (15/25)**	**20% (5/25)**	**20% (5/25)**

Table 2: Category-wise Success Rates Across AI Agents

and managing the training process. These tasks assess both technical skills and the ability to handle long-running tasks.

3. Debugging presents common scenarios including shape errors, exploding gradients, incorrect implementations, and integration errors. Agents must analyze large training logs, metrics, and code across multiple files to identify and resolve issues.

4. Model Implementation tests the ability to modify existing architectures and implement new features. An example is the ChannelViT related tasks, which follow three levels of increasing difficulty: Level 1 provides complete specifications with examples and tests; Level 2 includes specifications and tests but omits examples; Level 3 gives specifications but tests and examples are hidden

5. API Integration assesses the ability to work with essential ML development tools, particularly for logging and experiment tracking.

6. Performance optimization challenges agents to improve baseline implementations through iterative experimentation and hypothesis testing.

3.1 Evaluation Metrics

Tasks are evaluated based on binary success (✓) or failure ($\times$). The aggregate success rate for each agent is calculated as:

$$\text{Success Rate} = \frac{\text{Total Successful Tasks}}{\text{Total Tasks}} \times 100\% \tag{1}$$

Agents are assessed on their ability to complete tasks accurately without introducing errors or hallucinations.

4 Evaluation Framework

In this section we briefly describe the design of our evaluation framework, called Calipers, for running the benchmark. The framework consists of three components: agents, evaluation tasks, and metrics. Agents are evaluated on various Machine Learning tasks to generate metrics. We designed Calipers to allow easy addition of new evaluation tasks, agents and metrics, ensuring the benchmark can evolve alongside advances in ML development practices and tooling.

4.1 Evaluation Task

Each evaluation task consists of a task description, a set of input code and data files, and a validation logic which checks agent generated outputs and artifacts for correctness. Depending on the type of task, we implement various types of validation checks including

- Running tests on generated code to check for correctness
- Checking for the presence of all required output files and artifacts
- Evaluating agent generated model checkpoints for required performance
- Querying logged artifacts and metrics from wandb

4.2 Agents

Each agent is provided with two inputs in an evaluation run, the description of the task and a working directory populated with initial input files. The agent's outputs are task-specific artifacts which are saved in the working directory. These outputs are validated to determine success or failure. We generate the evaluation metrics discussed in the previous section for each evaluation run. We use litellm callbacks to capture metrics like number of steps, tokens, and cost.

5 Agent Setup

We evaluate three agents on ML-Dev-bench. The agents and their setup is described below. Each agent uses an LLM and a set of tools to execute various actions. All agents execute their code in a runtime environment which is either a local python or docker environment depending on the agent. We customized the runtime environments for all agents to pre-install common ML frameworks like scikit-learn, pytorch, transformers, lightning, wandb, etc to ensure smooth execution.

1. **ReAct:** We created a simple ReAct agent [8] as a baseline which takes actions by calling tools. We used the LangGraph framework for the agent and Composio toolset which provides tools for common use cases. We customized the tools to reliably capture large command outputs, handle long

running commands and ensure consistency across different tools like file and shell tools. All the tool calls were executed in a local python environment which was pre-installed with common ML frameworks as mentioned earlier and had access to the relevant api keys. No custom prompts were used, and the agent was allowed to run for a maximum of 50 steps. We tested the agent with Claude Sonnet 3.5 10-2022 and OpenAI GPT-4o.

   (a) **Command line tools**

      i. **Shell Tool -** to execute short running commands

      ii. **Spawn Tool -** to execute long running commands like training in the background

      iii. **Sleep and execute tool -** to wait and monitor long running processes

   (b) **File tools** like create files, list files and edit files

2. **Openhands:** Openhands [7] is a popular open-source coding agent with state-of-the-art performance on SWE-Bench-Full [5]. We used Openhands agent v0.21.1 and customized the runtime build to install common ML frameworks listed above. We tested the agent with Claude Sonnet 3.5 10-2022 model which is the current best performing model with the agent on SWE Bench. The agent was allowed to run for a maximum of 50 steps.

3. **AIDE:** AIDE is an agent purpose-built for data science tasks like Kaggle competitions [2] and performs a tree search over solutions. AIDE scaffolding performs better in comparison to other agents like Openhands on MLEBench using o1, GPT-4o. Unlike other general purpose agents which output any artifact, AIDE outputs an evaluation metric and code as its final output. All other artifacts are considered intermediate outputs and saved in a custom working directory. Since not all tasks in ML-Dev-Bench require outputting a score, we access the artifacts from its custom working directory to validate the agent's performance. Given the high costs of o1, we evaluated the agent with GPT-4o.

6 Performance Comparison

Performance of the agents across different task categories, Table 2 and individual tasks Table 3 reveals a consistent pattern. Performance decreases as tasks become more open-ended and complex. The success rates are highest in well-defined categories like dataset handling and basic debugging, but drop significantly in performance optimization tasks where no agent succeeded.

OpenHands-Sonnet (OH-Sonnet) and ReAct-Sonnet are the two best performing agents with 60% and 56% success rate respectively, while AIDE-4o and React-4o achieve 20% success rate. Both OH and ReAct Sonnet do well in structured tasks like dataset download, model training setup and debugging tasks with clear instructions but struggle in open-ended performance improvement and long-running training tasks.

Across the 14 common successful tasks, we did not observe any significant trends in cost across both OH-Sonnet and ReAct-Sonnet. While in long running tasks OH-Sonnet uses more tokens on average specially in debugging and training tasks, while ReAct shows higher usage in model implementation tasks like ChannelViT. Even in tasks with higher token usage, the costs dont scale proportionally due to efficient prompt caching in Openhands.

6.1 ReAct-Sonnet

ReAct-Sonnet achieved a success rate of 56% (14/25 tasks), demonstrating strong performance in specific, well-defined tasks but struggling with more complex scenarios. The agent excelled in dataset handling (3/3 tasks), basic model training (4/6 tasks), and debugging tasks with clear specifications (4/6 tasks). However, its performance degraded significantly in model implementation and performance optimization categories, where tasks required more open-ended problem-solving.

The agent's token usage and costs varied significantly across tasks. Simple operations like dataset downloads cost around $0.02 - 0.08$\$, while debugging tasks cost between $0.1 - 0.4$\$, some tasks like ChannelViT-Easy debugging take more steps indicating potentially inefficient exploration in complex scenarios.

A notable strength was ReAct-Sonnet's systematic approach to debugging when provided with specific instructions and test cases. However, the agent showed several limitations listed below:

1. Excessive verification seeking: The agent has a tendency to request feedback even when its instructed to run till completion, specially in complex tasks like model implementation and training.

2. Premature task termination in long-running training scenarios, where it doesnt wait until completion of long running tasks like training.

3. In long running tasks it fails to successfully implement sub-tasks which it handled correctly in insolation. For example, it correctly handles the noisy imagenette dataset setup in dedicated download tasks but fails the same operation when it's part of the longer training pipeline.

6.2 ReAct-4o

ReAct-4o had some success with tasks with well-defined specifications (WandB logging, downloading a specific model from Torchvision), and certain debugging tasks. However it did struggle on other tasks in the same categories. It also failed on the relatively easier tasks like dataset download due to not following instructions, ran into indentation errors while attempting to debug code and failing to produce output artifacts as required by certain tasks.

6.3 OpenHands-Sonnet

OpenHands-Sonnet demonstrated the highest success rate at 60% (15/25 tasks), showing robust performance across most categories. The agent successfully completed all dataset handling tasks (3/3) and showed strong performance in model training (5/6) and debugging (4/6).

The agent particularly excelled in structured tasks and showed better persistence in long-running operations. However, it struggled with performance optimization tasks (0/5), indicating limitations in open-ended problem-solving scenarios requiring iterative improvement.

6.4 Aide-4o

Aide-4o had a 20% success rate (5/25 tasks), demonstrating limitations across most categories. The agent managed to complete some basic dataset handling and debugging tasks but struggled with model training (2/6) and completely failed in model implementation and performance optimization categories.

The cost metrics for Aide-4o weren't captured as it doesnt use LiteLLM, but its low success rate across both specific and open-ended tasks suggests limitations in handling ML development workflows. The agent's successes were primarily limited to tasks with very clear, step-by-step instructions and immediate feedback loops.

7 Conclusion

We presented ML-Dev-Bench, a benchmark focused on ML development workflows consisting of 25 tasks. We evaluated 3 agents on this benchmark - ReAct (with Claude Sonnet and GPT-4o), Openhands and AIDE; Openhands with Claude Sonnet performed the best out of these. Future work can involve analysing the impact of scaling compute on these agents; computing variance in success metrics across multiple runs; including reasoning models such DeepSeek-R1, O-1/O-3; and expanding the problem categories to include areas such as label collection. We open-source the evaluation framework for the benefit of the broader community.

References

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

[2] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2024.

[3] Jeremy Howard and Others. Imagenette. `https://github.com/fastai/imagenette`, 2019.

[4] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation, 2024.

[5] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024.

[6] Xiangru Tang, Yuliang Liu, Zefan Cai, Yanjun Shao, Junjie Lu, Yichi Zhang, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Liang Chen, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yin Fang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and Mark Gerstein. Ml-bench: Evaluating large language models and agents for machine learning tasks on repository-level code, 2024.

[7] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2024.

[8] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

Task	**Category**	**ReAct-Sonnet**	**OH-Sonnet**	**Aide-4o**	**ReAct-4o**
Dataset download - Noisy Imagenette	Dataset Handling	✓	✓	×	×
Dataset download - dataset does not exist	Dataset Handling	✓	✓	✓	×
Dataset preprocessing	Dataset Handling	✓	✓	×	×
Pretrained model download - Torchvision	Model Training	✓	✓	✓	✓
Pretrained model download - HuggingFace	Model Training	✓	✓	×	×
Vision finetuning - classification	Model Training	✓	✓	×	×
Overfit on small dataset	Model Training	✓	✓	✓	✓
Large training logs	Model Training	×	✓	×	✓
CIFAR10 Training	Model Training	×	×	×	×
Fix problems in model and dataloader	Debugging	×	×	×	×
Model forward pass - shape mismatches	Debugging	✓	✓	✓	✓
Model Training - shape mismatches	Debugging	✓	✓	✓	×
NaN losses	Debugging	✓	✓	×	×
Correct norm for pretrained model	Debugging	✓	✓	×	×
TinyBERT Eval	Debugging	×	×	×	×
Wandb integration	API Integration	✓	✓	×	✓
ChannelViT - Easy	Model Implementation	✓	✓	×	×
ChannelViT	Model Implementation	✓	✓	×	×
ChannelViT - No tests	Model Implementation	×	×	×	×
VAR implementation	Model Implementation	×	×	×	×
Improve CIFAR-10 baseline - existing model ckpt	Performance	×	×	×	×
Noisy Imagenette	Performance	×	×	×	×
CIFAR-10 long tailed	Performance	×	×	×	×
Segmentation	Performance	×	×	×	×
BoolQ	Performance	×	×	×	×
Success Rate		**56% (14/25)**	**60% (15/25)**	**20% (5/25)**	**20% (5/25)**

Table 3: Performance Comparison Across AI Agents

Task	**Token Cost ($)**		**Total Tokens**	
	ReAct	**OH**	**ReAct**	**OH**
Vision finetuning - classification	0.176	0.124	67,034	54,434
ChannelViT	1.06	0.215	338,055	177,182
ChannelViT - Easy	1.090	0.318	352,141	267,508
ChannelViT - No tests	0.091	0.121	33,208	53,475
Dataset download - dataset does not exist	0.018	0.051	13,735	16,125
Dataset preprocessing	0.078	0.103	27,210	42,277
Model forward pass - shape mismatches	0.069	0.075	27,629	44,396
Pretrained model download - HuggingFace	0.096	0.063	36,592	24,738
CIFAR-10 long tailed	0.089	0.334	29,659	351,943
Fix problems in model and dataloader	0.376	0.556	146,826	785,182
NaN losses	0.124	0.212	40,385	193,801
Dataset download - Noisy Imagenette	0.129	0.068	67,426	25,429
Correct norm for pretrained model	0.380	0.265	136,831	313,149
Overfit on small dataset	0.093	0.133	25,642	52,791
Large training logs	0.023	0.185	35,189	114,903
Segmentation	0.118	0.383	39,662	395,127
Pretrained model download - Torchvision	0.058	0.044	22,289	30,596
CIFAR10 Training	0.209	0.288	71,409	253,445
Model Training - shape mismatches	0.289	0.147	115,568	92,262
Add implementation - VAR	0.051	0.139	12,473	59,863
Wandb integration	0.155	0.258	65,810	266,738
TinyBERT Eval	0.313	0.573	121,773	762,780
BoolQ	0.343	0.650	131,458	993,175
Improve CIFAR-10 baseline - existing model ckpt	0.115	0.401	29,232	366,660
Noisy Imagenette	0.582	0.288	192,275	253,445

Table 4: Comparison of Token Metrics between ReAct-Sonnet and OpenHands-Sonnet