LLM Agents Struggle at Engineering Time Series Solutions

Anonymous Authors[1]

Abstract

Large Language Model (LLM) agents are increasingly used for machine learning (ML) research and engineering tasks, but how well do they handle time series challenges? The results of our investigation are not optimistic. The application of agentic AI in support of time series analytics is not in a mature state yet, and its performance is not evaluated comprehensively and thoroughly enough to inspire confidence for real-world applications. Existing benchmarks lack scalability, focus narrowly on model building in idealistic, well-defined settings, and evaluate only a limited set of research artifacts (such as CSV result files often submitted to Kaggle competitions), coming short of assessing other pragmatic aspects of competency of the agentic tools, such as, e.g., data wrangling abilities. Effective ML engineering, whether human- or AI-driven, requires a broad set of diverse skills to competently approach challenges commonly encountered in practice in order to deliver complete solutions. Our experiments demonstrate how state-of-the-art agents struggle to solve time series ML engineering tasks, and how current benchmarks do not challenge them well enough. We argue that our community still needs more competent agents and more comprehensive benchmarks to produce ML engineering LLM-driven agents capable of solving real world time series challenges.

1. Introduction

Large Language Model (LLM) agents have shown growing promise in reducing the mundane, mostly manual efforts required in machine learning (ML) engineering and improving the overall productivity of ML practice. Several benchmarks have been introduced to evaluate the capabilities of LLM agents in ML tasks (Tab. 1). However, these benchmarks have important limitations. First, many of them source ML challenges primarily from Kaggle, which are well-structured and do not fully capture the complexity of real-world ML tasks. Second, evaluations are typically outcome-based, focusing on overall task completion or downstream model performance metrics such as accuracy. These metrics combine and obscure the impact of multiple skills that jointly determine the results, such as effective data wrangling or code quality improvement capabilities. Third, current benchmarks lack scalability, as tasks have to be manually curated and cannot be developed at scale.

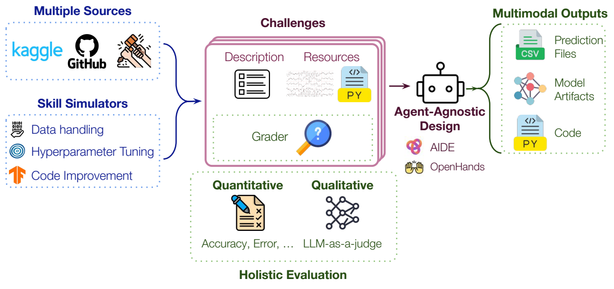

Figure 1. `TimeSeriesGym` is a scalable benchmarking environment for ML engineering agents. It features 34 time series challenges across 8 unique time series problems, spanning more than 15 domains. Challenges are either carefully designed based on real-world ML practice, or sourced from Kaggle competitions and GitHub repositories. `TimeSeriesGym` enables efficient and scalable generation of new challenges. Our evaluation methodology combines precise quantitative metrics with flexible qualitative assessment, and provides specialized tools to grade various artifacts generated during ML engineering. `TimeSeriesGym` is compatible with many different agent types.

To address these limitations, we introduce `TimeSeriesGym`, a comprehensive benchmarking framework designed to evaluate LLM agents on time series ML engineering tasks (Fig. 1). This framework is both scalable and agent-agnostic, incorporating traditional Kaggle-style competitions alongside carefully crafted tasks that reflect real-world ML engineering practices. We focus specifically on time series data for two key reasons. First, time series represents one of the most common

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Table 1. Comparison of `TimeSeriesGym` with existing ML/DS agent benchmarks. Categories include **Number:** total and time series (TS) tasks in each benchmark, where each task corresponds to a unique data source (e.g., a single Kaggle competition or GitHub repository); **Source:** task origins (K: Kaggle, G: GitHub, H: Hand-crafted); **ML Capability:** coverage of ML **science** tasks (e.g., modeling, open-ended research) and **engineering** tasks (e.g., repository utilization, API integration); and **Evaluation:** capabilities for evaluating **multimodal** outputs (e.g., prediction files, model artifacts), specific ML **skills** (e.g., data handling, model improvement), and from a **holistic** perspective (combining quantitative metrics (accuracy, mean absolute error) with qualitative evaluation via code landmarks or LLM-as-a-judge approaches). We use "+" to indicate `TimeSeriesGym`'s scalability which enables the generation of an unlimited number of new challenges using the tools provided.

	Number		**Source**			**ML Capability**		**Evaluation**		
	Total	TS	K	G	H	Science	Engineering	Multimodal	Skill-based	Holistic
MLE-bench (Chan et al., 2025)	75	3	✓	✗	✗	✓	✗	✗	✗	✗
MLAgentBench (Huang et al., 2024)	13	1	✓	✗	✓	✓	✗	✓	✗	✗
MLGym (Nathani et al., 2025)	13	0	✓	✗	✓	✓	✗	✓	✗	✗
RE-Bench (Wijk et al., 2024)	7	0	✗	✗	✓	✓	✗	✓	✗	✗
DSBench[1] (Jing et al., 2025)	74	5	✓	✗	✗	✓	✗	✗	✗	✗
SUPER[2] (Bogin et al., 2024)	45	0	✗	✓	✗	✗	✓	✗	✗	✓
ML-Bench (Tang et al., 2023)	18	1	✗	✓	✗	✗	✓	✗	✗	✗
ML-Dev-Bench (Padigela et al., 2025)	30	0	✗	✗	✓	✗	✓	✓	✓	✗
`TimeSeriesGym` (Ours)	23+	23+	✓	✓	✓	✓	✓	✓	✓	✓

forms of structured data in practical applications. Second, this focus allows us to examine how foundation models handle structured information processing, with agents serving as the primary interface through which LLMs interact with and manipulate such data. Our experimental results reveal significant limitations in current technology. Even the most advanced open-source agents demonstrate poor performance on time series ML engineering tasks, highlighting fundamental gaps in how existing foundation models process and work with structured data.

2. `TimeSeriesGym` and Problems with Existing Benchmarks

We envision `TimeSeriesGym` as a scalable benchmarking environment for time series machine learning engineering. The current version features 34 challenges from 23 unique data sources across 8 unique time series problems, spanning more than 15 domains. These challenges evaluate LLM agents on a range of realistic ML engineering skills beyond just model development, including data labeling, model selection, and the utilization, improvement, and migration of research code (Tab. 3).

Each challenge in `TimeSeriesGym` is organized with a consistent structure: (1) **resources** including datasets, code repositories, related paper(s) and documentation relevant to the challenge; (2) a **description file** that outlines the challenge parameters, available resources, and provides specific instructions and hints for successful completion; and (3) **challenge-specific grading functions** to evaluate agent submissions. Some challenges also include leaderboards to rank agent submissions against human performance. These leaderboards are readily available for, e.g., challenges derived from Kaggle competitions.

Challenges are derived from Kaggle competitions (currently, n = 12) and popular benchmarks and research code repositories for time series modeling (`TimeSeriesGym` Originals, n = 14). Each challenge is specifically chosen or designed to evaluate one or more of the following skills: (1) *Data Handling:* Ability to handle missing data, use data labeling tools, and leverage multi-source data for model building. (2) *Modeling:* Ability to develop useful time-series ML models, tune hyperparameters, perform model selection, and understand, utilize, migrate and improve the quality of research code. (3) *Benchmarking:* Training and rigorously evaluating ML models using standard benchmarks. We prioritized challenges that reflect core skills that are regularly exercised by ML engineers, researchers, and data scientists; and have broad coverage across diverse domains (e.g., healthcare, finance, epidemiology) and time series problems (forecasting, classification, time series understanding). Tab. 3 provides a comprehensive overview of each challenge within `TimeSeriesGym`, including its domain, core problem, evaluation metric, and the skills required to address it.

ML engineering benchmarks can be resource-intensive. For instance, experiments on a single seed for `MLE-Bench` cost approximately USD 2500. To improve accessibility of these benchmarks, we propose `TimeSeriesGym-Lite`, a carefully selected subset of six challenges designed to efficiently evaluate LLM agents on critical ML engineering skills while maintaining coverage across multiple domains

[1]For DSBench, we include only data modeling tasks, while excluding data analysis tasks as they are not directly relevant to our work. For SUPER, we include repositories used to create the Expert and Masked sets of the benchmark.

and time series problems. This collection enables rapid and cost-effective assessment of novel LLM agents without sacrificing the diversity of skills being tested (see Tab. 5).

Generating challenges at scale. We provide key mechanisms to enable efficient and scalable generation of new challenges. First, we offer clear and detailed documentation that explains how to add new challenges to the benchmark. Second, we provide specialized tools to create skill-specific challenges (e.g., simulating missing data) and evaluate them. Using these resources, our team successfully created several new challenges in two hours of effort, each testing specific ML engineering skills. This scalability ensures that `TimeSeriesGym` can grow and adapt as time series machine learning techniques continue to advance.

Multimodal, skill-based, holistic evaluation. Existing benchmarks typically summarize agent performance using metrics such as accuracy, completion rate, or competition rankings (Chan et al., 2025). Although these metrics provide useful summaries, they do not offer much actionable feedback for improvement. `TimeSeriesGym` addresses this limitation through an evaluation framework designed to provide specific actionable feedback through multiple complementary approaches. First, we design challenges that isolate and test specific skills, such as handling missing data (e.g., `Optiver Realized Volatility Prediction with Missing Data`). Poor performance on these targeted challenges clearly indicates potential skill gaps, enabling developers to focus their efforts on specific skills. Second, we develop fine-grained evaluation tools that assess multiple dimensions of performance simultaneously. For example, in code migration tasks (e.g., `Convert ResNet from TensorFlow to PyTorch`), our evaluation tools examine whether an agent follows instructions and naming conventions, completes all required function definitions, in addition to successful execution– providing a multidimensional performance profile rather than a binary success/failure indicator.

Our evaluation methodology deliberately combines multiple assessment approaches: quantitative metrics (accuracy, mean absolute error), programmatic analysis (regular expression matching, code inspection), and qualitative evaluation (LLM-as-a-judge) (see Appendix F). This hybrid approach balances the reliability of objective metrics with the flexibility of subjective assessment. Although LLM-based evaluation offers valuable insight, especially for open-ended tasks such as research code enhancement, we recognize that LLMs can be inconsistent and prone to hallucination. To mitigate these concerns, we strategically complement subjective assessments with precise quantitative metrics, creating a robust evaluation system that leverages the strengths of each approach while offsetting their individual limitations.

Furthermore, `TimeSeriesGym` provides specialized tools to grade diverse artifacts generated throughout the ML engineering lifecycle– from submission files (`CSV`, `H5`, etc.) to source code (`.py`) and trained models (`.pth`, `.pkl`)– enabling comprehensive assessment of the entire development process rather than focusing solely on final outputs.

3. Experiments and Results

Table 2. **Main Results**. Each experiment was run with 3 random seeds, with results showing mean ± standard deviation. The table compares scaffold types (`OpenHands` vs. `AIDE`), model choices (`GPT-4.1`, `o3`, `Claude 3.7`), resource allocations (4/50 to 12/150 hours/steps), and time utilization approaches. Key findings include: (1) `AIDE` outperforms `OpenHands` as a scaffold, (2) the reasoning model `o3` achieves significantly higher valid submission rates (94.4%) than other models, and (3) `Claude 3.7` produces the most reasonable submissions (38.9%).

Lite	**Model**	**Resources** (hours / steps)	**Valid Submission** (%)	**Reasonable Submission** (%)
	`OpenHands`			
✓	+ `gpt-4.1-2025-04-14`	4 / 50	44.4 ± 19.3	11.1 ± 9.6
	`AIDE`			
✗	+ `gpt-4.1-2025-04-14`	4 / 50	57.3 ± 7.9	12.5 ± 0
✓	+ `gpt-4.1-2025-04-14`	4 / 50	66.7 ± 16.7	27.8 ± 9.6
✓	+ `o3-2025-04-16`	4 / 50	**94.4 ± 9.6**	33.3 ± 0.0
✓	+ `claude-3-7-sonnet-20250219`	4 / 50	50.0 ± 16.7	38.9 ± 19.3
Effect of Scaling Resources				
✓	+ `gpt-4.1-2025-04-14`	4 / 50	66.7 ± 16.7	27.8 ± 9.6
✓	+ `gpt-4.1-2025-04-14`	8 / 100	72.2 ± 9.6	22.2 ± 9.6
✓	+ `gpt-4.1-2025-04-14`	12 / 150	61.1 ± 9.6	**50.0 ± 0.0**

Setting. We run agents in an Ubuntu 20.04 Docker container with all necessary resources (datasets, code repositories, etc.) and basic Python packages useful for ML engineering. Agents can access the internet and install additional packages as needed. For each challenge, agents have a maximum of 4 hours and 50 steps (Huang et al., 2024; Padigela et al., 2025; Nathani et al., 2025) and use a machine with 128 vCPUs, 503 GB RAM, 1.8 TiB SSD, and a single NVIDIA A100-SXM4-80GB GPU[3]. Unless otherwise specified, we repeat each experiment with 3 different seeds (0, 1, and 2) to calculate mean and standard deviation.

Metrics. We report the raw scores achieved by each agent on every challenge (Tab 3). Although these scores are useful for tracking progress on individual challenges, they cannot be easily combined across different challenges. To measure the performance of agents at a high level, we report two key metrics: the percentage of challenges where the agent made a (1) *valid*, and (2) *reasonable* submission. A submission is valid if the grader returns any non-null score. What counts as a reasonable attempt varies by challenge type (Tab. 2). For Kaggle challenges, we define a reasonable

[3]In practice, agents share this machine as we run multiple challenges in parallel. This represents a realistic setting similar to how ML engineers routinely share computing resources. We found no instances where this sharing might have disadvantaged any agent.

attempt as scoring above median on the competition's public leaderboard. For the remaining challenges, a reasonable submission shows a genuine attempt at generating a valid submission, rather than hallucinating an output that matches the submission format. We determine this by examining agent logs either manually or using LLM-as-a-judge.

3.1. Observations

`AIDE` is the better open-source scaffold. We evaluated `GPT-4.1` (`gpt-4.1-2025-04-14`) using two open-source scaffolds: `AIDE` (Jiang et al., 2025) and `OpenHands` (Wang et al., 2024). Following `MLE-bench`, we make minor modifications to adapt these scaffolds to our benchmark (see Appendix C). Our results in Tab. 2 confirm previous findings: `GPT-4.1` produces more valid (66.7% vs 44.4%) and reasonable (27.8% vs 11.1%) submissions with `AIDE` than with `OpenHands`. This is expected because `AIDE` is specifically designed for data science tasks, which account for the majority of the `TimeSeriesGym` challenges.

Reasoning models produce substantially more valid submissions. To identify the best base model, we conducted experiments using the best scaffold (`AIDE`) with two state-of-the-art proprietary LLMs: `GPT-4.1` (`gpt-4.1-2025-04-14`) and `Claude 3.7 Sonnet` (`claude-3-7-sonnet-20250219`), and a reasoning model `o3` (`o3-2025-04-16`). As shown in Tab. 2, our experiments on `TimeSeriesGym-Lite` revealed that `o3` created significantly more valid submissions than other models, while `Claude 3.7` produced the highest number of reasonable attempts (38.9%).

Challenges in `TimeSeriesGym` are hard for state-of-the-art agents. We tested `AIDE` with `GPT-4.1` on all `TimeSeriesGym` challenges and found poor overall performance. The agent produced valid submissions for only 57.3% of challenges and reasonable submissions for just 12.5%. We found that this agent especially struggled with `TimeSeriesGym` original challenges, where it failed to produce valid submissions for 5 out of 13 challenges (Tab. 8). These results show that even the best agents struggle with ML engineering tasks, particularly those that go beyond standard Kaggle data science challenges and involve working with multi-file code repositories.

Agents do not improve with more time. We wondered if the agents perform poorly on `TimeSeriesGym` simply because they need more time. To test this idea, we ran `AIDE` with `GPT-4.1` on `TimeSeriesGym -Lite` and gave it 2 or 3 times more hours and steps to solve each challenge. Our results show that extra time does not always improve performance (Tab. 9). Even with the maximum time (12 hours and 150 steps), the agent only made reasonable submissions in about 50% of challenges. Although this might seem promising, it is not very impressive, since the bar for a "reasonable" submission is quite low.

4. Open Questions and Opportunities

Key limitations of existing scaffolds. Agentic scaffolds such as `AIDE` and `OpenHands` provide structured workflows that **excel in single-shot, self-contained benchmarks** (e.g., Kaggle competitions) but reveal **significant limitations in repository-level challenges** that require multiple file edits and iterative reasoning. `AIDE`'s *one-step solution strategy* and *fixed action set*—restricted to predefined operations such as "data preview" when debugging—often lead to unsuccessful attempts in large codebases, as the agent's attention is diluted across irrelevant files and fails to identify critical information. Conversely, `OpenHands` supports multi-step trajectories yet suffers from a **greedy exploitation bias**: it commits fully to a single approach without exploring alternative solution paths or revisiting earlier decisions when trajectories prove unfruitful. The planning algorithm of the `CodeAct` agent used by `OpenHands` is similarly *greedy and short-horizon*, limiting adaptation to complex multistage development workflows. These findings highlight the need for **more adaptive scaffolds** that dynamically expand their action repertoire, balance exploration and exploitation through parallel solution threads, and support nested workflows reflective of real-world machine learning engineering tasks. We provide illustrations of agent failures in Appendix E.

Optimal resource allocation. Consistent with previous work, agents were given 4 hours and 50 steps to solve each challenge - but is this sufficient? Alternative frameworks like `MLE-bench` provide substantially more resources (24 hours and approximately 2000 steps). Our scaling experiments, which gave agents up to 12 hours and 150 steps for a subset of challenges, did not reveal significant performance improvements. Therefore, we believe that further increasing resources is an option, but practical academic budget constraints make such approaches largely infeasible. This raises important questions about how to balance resource limitations with fair opportunities to assess LLM agents.

5. Conclusion

We propose `TimeSeriesGym`, a scalable and agent-agnostic benchmarking framework to evaluate LLM agents on ML engineering tasks in time series. We show that while frontier LLMs combined with `AIDE` scaffolding (Jiang et al., 2025) can achieve moderate to high success rates in producing valid submissions, they still do not generate reasonable solutions, particularly on `TimeSeriesGym-Originals` that emulate the complexity of real-world time series tasks.

Acknowledgements

Omitted for double blind review.

Impact Statement

Societal Impact. AI agents promise to substantially reduce manual effort in ML engineering while expanding the productivity and accessibility of ML tools. This automation presents several social implications worth considering. First, by lowering technical barriers, these agents could democratize ML capabilities, allowing users without an extensive programming background to leverage advanced analytics. Second, automated ML workflows can accelerate scientific discovery in multiple application domains, including healthcare, climate science, and materials research. However, several challenges require careful attention from the community. The primary concern is proper attribution when agents repurpose existing code, potentially obscuring original authorship and violating licenses. Furthermore, automated ML systems can perpetuate or amplify existing biases in training data without human oversight. There is also the risk of workforce displacement for entry-level ML engineers as routine tasks become increasingly automated. Furthermore, these agents might generate plausible but flawed solutions that appear correct to non-experts, leading to undetected errors in critical applications. The resource-intensive nature of running sophisticated agents could also exacerbate computational divides between well-resourced and under-resourced organizations. As we advance agent capabilities through benchmarks like `TimeSeriesGym`, the research community must simultaneously develop frameworks for responsible deployment that address these challenges while maximizing societal benefits.

Data leakage and plagiarism. In designing `TimeSeriesGym`, we identify two key risks related to data leakage and plagiarism that could compromise the integrity of the benchmark: (1) **Pretraining contamination:** Current LLMs may have been exposed to public content from existing challenges (e.g., Kaggle competitions), including task descriptions, data, or shared solutions. This can lead to memorization and inflated performance that overstates agents' true capabilities, and (2) **Future LLM leakage:** Once the benchmark is public, future LLMs may be pretrained on its content, making the benchmark less effective in evaluating real generalization.

To address such risks, we present both empirical findings and mitigation strategies. For case (1), we have two key observations. First, in both Kaggle-based and original challenges in `TimeSeriesGym`, agents either performed poorly or did not produce valid output, suggesting minimal benefit from any potential LLM contamination. Second, we conducted a formal analysis using available tools to assess agents' familiarity with all competitions in this benchmark. The results show no evidence of systematic prior exposure or memorization, further supporting the integrity of the benchmark in its current state. For case (2), the scalability of `TimeSeriesGym` enables efficient generation of new challenges and skill-specific variations. This allows the benchmark to evolve continuously and remain effective even if the current version is eventually included in future LLM pretraining.

Finally, we raise a broader question around plagiarism and code reuse. Several `TimeSeriesGym` challenges, such as leveraging `MOMENT` (Goswami et al., 2024) for anomaly detection, require agents to use existing code repositories to solve open-ended ML tasks. In such cases, it becomes difficult to clearly define and assess plagiarism. For example, if an agent cites the code it uses, should it be considered plagiarism or appropriate reuse, similar to how human ML practitioners build on public code with proper reference? As the ability to effectively and properly leverage existing resources is important in real-world ML practice, we believe that it is crucial to develop clear, legally correct definitions and evaluation criteria for data contamination and plagiarism in the context of LLM agents. We highlight this as an important direction for future work.

Reproducibility statement

Upon acceptance, we will provide `TimeSeriesGym` as an open-source project under the permissive MIT License. The repository will include detailed documentation on running experiments, adding new challenges, and incorporating different agentic scaffolds. Tab. 3 lists all challenges in `TimeSeriesGym`, while Tab 4 provides their sources and licenses. We describe our exact experimental settings and compute resources in Sec. 3, with scaffold hyperparameters detailed in Tab. 7. The cost to run each experiment is reported in Tab. 6.

References

Bogin, B., Yang, K., Gupta, S., Richardson, K., Bransom, E., Clark, P., Sabharwal, A., and Khot, T. SUPER: Evaluating agents on setting up and executing tasks from research repositories. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 12622–12645, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.702. URL https://aclanthology.org/2024.emnlp-main.702/.

Cai, Y., Choudhry, A., Goswami, M., and Dubrawski, A. TimeSeriesExam: A time series understanding exam.

NeurIPS'24 Time Series in the Age of Large Models Workshop, 2024.

Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Madry, A., and Weng, L. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

Embretson, S. E. and Reise, S. P. *Item response theory for psychologists*. Psychology Press, 2013.

Goswami, M., Sanil, V., Choudhry, A., Srinivasan, A., Udompanyawit, C., and Dubrawski, A. Aqua: A benchmarking tool for label quality assessment. *Advances in Neural Information Processing Systems*, 36:79792–79807, 2023.

Goswami, M., Szafer, K., Choudhry, A., Cai, Y., Li, S., and Dubrawski, A. MOMENT: A family of open time-series foundation models. In *International Conference on Machine Learning*, pp. 16115–16152. PMLR, 2024.

Guinet, G., Omidvar-Tehrani, B., Deoras, A., and Callot, L. Automated evaluation of retrieval-augmented language models with task-specific exam generation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=4jqOV6NlUz.

Ho, C., Zou, J., Alama, O., Jagadesh Kumar, S. M., Chiang, C.-Y., Gupta, T., Wang, C., Keetha, N., Sycara, K., and Scherer, S. Map it anywhere: Empowering bev map prediction using large-scale public datasets. *Advances in Neural Information Processing Systems*, 37:64433–64453, 2024.

Huang, Q., Vora, J., Liang, P., and Leskovec, J. MLAgentbench: Evaluating language agents on machine learning experimentation. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=1Fs1LvjYQW.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Jing, L., Huang, Z., Wang, X., Yao, W., Yu, W., Ma, K., Zhang, H., Du, X., and Yu, D. DSBench: How far are data science agents from becoming data science experts? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=DSsSPr0RZJ.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., and Zhu, C. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, 2023.

Massey Jr, F. J. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46 (253):68–78, 1951.

Mialon, G., Fourrier, C., Wolf, T., LeCun, Y., and Scialom, T. GAIA: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

Nathani, D., Madaan, L., Roberts, N., Bashlykov, N., Menon, A., Moens, V., Budhiraja, A., Magka, D., Vorotilov, V., Chaurasia, G., Hupkes, D., Cabral, R. S., Shavrina, T., Foerster, J., Bachrach, Y., Wang, W. Y., and Raileanu, R. MLGym: A new framework and benchmark for advancing ai research agents, 2025. URL https://arxiv.org/abs/2502.14499.

Padigela, H., Shah, C., and Juyal, D. ML-Dev-Bench: Comparative analysis of ai agents on ml development workflows. *arXiv preprint arXiv:2502.00964*, 2025.

Risdal, M. and Bozsolik, T. Meta kaggle, 2022. URL https://www.kaggle.com/ds/9.

Tang, X., Liu, Y., Cai, Z., Shao, Y., Lu, J., Zhang, Y., Deng, Z., Hu, H., An, K., Huang, R., et al. ML-Bench: Evaluating large language models and agents for machine learning tasks on repository-level code. *arXiv preprint arXiv:2311.09835*, 2023.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., et al. Openhands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2024.

Wijk, H., Lin, T., Becker, J., Jawhar, S., Parikh, N., Broadley, T., Chan, L., Chen, M., Clymer, J., Dhyani, J., et al. Rebench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

Ye, W., Zhang, Y., Yang, W., Tang, L., Cao, D., Cai, J., and Liu, Y. Beyond forecasting: Compositional time series reasoning for end-to-end task execution. *arXiv preprint arXiv:2410.04047*, 2024.

A. Related Work

Machine learning agent benchmarks. Several benchmarks have been proposed to evaluate LLM agents on automating ML and DS tasks. These benchmarks are typically structured around three key components: (1) task curation, (2) agent capabilities being evaluated, and (3) evaluation protocol. Benchmarks differ in how they curate ML/DS tasks. For example, MLE-bench (Chan et al., 2025) and DSBench (Jing et al., 2025) compile tasks from online competition platforms such as Kaggle, while other benchmarks source tasks from ML-related Github repositories (Bogin et al., 2024; Tang et al., 2023) or hand-craft tasks based on ML research problems or engineering workflows (Huang et al., 2024; Nathani et al., 2025; Wijk et al., 2024; Padigela et al., 2025). With regard to agent capabilities, some benchmarks (Chan et al., 2025; Huang et al., 2024; Nathani et al., 2025; Wijk et al., 2024; Jing et al., 2025) focus on comprehensive ML science skills by evaluating agents on end-to-end problem solving skills, while others (Bogin et al., 2024; Tang et al., 2023; Padigela et al., 2025) focus on more modular engineering-oriented capabilities within the ML pipeline, such as using GitHub repositories or integrating APIs. Evaluation protocols also differ in output formats and granularity. MLE-bench (Chan et al., 2025) and DSBench (Jing et al., 2025) require agents to output results in specific formats (e.g., CSV files) that can be directly scored using predefined metrics such as accuracy, while other benchmarks (Huang et al., 2024; Nathani et al., 2025; Wijk et al., 2024) allow for more flexible outputs in addition to prediction files, such as model artifacts and code. ML-Dev-Bench (Padigela et al., 2025) further extends the evaluation by specific skills (e.g., data handling, model improvement), while SUPER (Bogin et al., 2024) provides a more holistic evaluation by combining outcome-based evaluation with qualitative code inspection to assess agents' progress towards completing the tasks.

Scalable dynamic benchmarks and holistic evaluation. Scalable benchmarks reduce manual data curation efforts by generating target problems at scale using carefully designed templates (Cai et al., 2024; Ye et al., 2024) or data engines (Ho et al., 2024), among which TimeSeriesExam (Cai et al., 2024) further improves problem sample quality by applying Item Response Theory (IRT) (Embretson & Reise, 2013; Guinet et al., 2024) to intelligently select questions with contextualized difficulty and appropriate discrimination. To remain effective against data contamination from LLM pretraining, dynamic benchmarks such as GAIA (Mialon et al., 2023) and LiveCodeBench (Jain et al., 2024) propose to continually incorporate problems newly released after LLM training cut-offs. While most benchmarks target specific capabilities, holistic evaluation (Liang et al., 2022; Goswami et al., 2023) provides a comprehensive picture through evaluating models on a wide range of datasets and tasks across diverse domains using multiple complementary metrics, to capture both the breadth and depth of model capabilities.

B. `TimeSeriesGym` Challenges

B.1. Curating Challenges

To identify Kaggle challenges for inclusion in the `TimeSeriesGym`, we began with the Meta Kaggle dataset (Risdal & Bozsolik, 2022), focusing specifically on Featured and Research competitions. Featured competitions are real-world ML challenges that pose difficult, commercially oriented prediction problems, while Research competitions offer opportunities to work on problems that may not have clean or straightforward solutions[4]. We employed `Gemini 2.0 Flash` to analyze competition descriptions and titles, identifying 453 competitions that likely involve time series data. Subsequently, these were ranked based on three key metrics: participant count, maximum reward offered, and presence of a public leaderboard. From the resulting shortlist of 100 high-quality competitions, we made our final selections to ensure comprehensive coverage across diverse tasks and domains within the time series analytics landscape.

To complement the selected Kaggle challenges, we include 14 `TimeSeriesGym` Original challenges, manually curated based on recommendations from experienced ML engineers and researchers. These challenges are specifically designed to evaluate advanced technical skills that Kaggle competitions typically cannot easily assess, yet are essential for effective ML engineering. Examples include utilizing state-of-the-art models (e.g., `Implement the MOMENT (Goswami et al., 2024) time series foundation model for anomaly detection`), migrating frameworks (e.g., `Convert ResNet-1D classification models from TensorFlow to PyTorch`), and improving research code quality (e.g., `Improve PTB-XL ECG Classification Code`. These capabilities represent critical competencies of skilled ML engineers that extend beyond the scope of standard Kaggle-like competitions.

[4]`https://www.kaggle.com/docs/competitions`

B.2. Design Choices

Focus on time series tasks. We focused on time series modeling tasks for two key reasons. First, time-series data are ubiquitous, and effectively modeling them can significantly advance critical domains such as healthcare and economics. Despite this importance, existing agentic AI benchmarks include relatively few time series challenges (Tab. 1). Second, compared to text and images, time series data require modest resources for storage and modeling, making `TimeSeriesGym` efficient to run. Moreover, modeling time-series data remains relatively underexplored outside specialized research communities, meaning that LLMs are less likely to have encountered such data and tasks during training. This characteristic, combined with the fact that `TimeSeriesGym` evaluates general machine learning skills, makes it an excellent testbed to evaluate AI agent capabilities. Moreover, the tools in this benchmark can be easily used to include non-time-series problems as well.

How much freedom should the agents be given? When designing challenges for `TimeSeriesGym`, we had to strike a fine balance between giving agents freedom to solve problems creatively while keeping enough structure in place to allow for a precise and fine-grained evaluation. For example, in the `PTB-XL ECG Classification with Hyper-parameter Optimization` challenge, we *required* agents to use a PyTorch-based neural network and save their models, files and code before and after tuning. This allowed us to inspect models and code to check if the hyperparameters changed, and measure how these changes improved performance.

Agent-agnostic design. `TimeSeriesGym` is agnostic to specific agent implementations. Following `MLE-bench` (Chan et al., 2025), it is easy to add new challenges and agentic scaffolds. To illustrate this flexibility, we include *latest* implementations of 3 different scaffolds, `AIDE` (Jiang et al., 2025), `ResearchAgent` (Huang et al., 2024), and `OpenHands` (Wang et al., 2024) with fundamentally different designs. Unlike `MLGym` (Nathani et al., 2025), we do not advocate for a default agentic scaffold, as we believe that agent designs will continue to evolve and no single scaffold will work best for all ML engineering tasks.

C. Implementation Details for Scaffolds

Table 7 summarizes the hyperparameters used for the two scaffolds: `AIDE` (Jiang et al., 2025) and `OpenHands`' `CodeAct` (Wang et al., 2024). We did not directly use the MLE-bench (Chan et al., 2025) modifications to the agentic scaffolds for two main reasons. First, the official scaffold implementation has undergone updates since MLE-bench. Second, certain changes made in MLE-bench were not applicable to our benchmark, which involves more diverse modes of evaluation. Therefore, we modified the latest version of the agentic scaffolds to enhance robustness, improve execution stability, and support a broader range of competitions.

C.1. AIDE

We forked the original `AIDE` repository in May 2025 and added useful changes from the MLE-bench project. These include better API calls and support for more API providers. Our key modifications are summarized below:

1. Updated instructions to cover all types of tasks, not just Kaggle challenges
2. Modified the prompt to work with different file types, since our tasks use many input/output formats
3. Fixed the interpreter handling to prevent timeouts and system hangs

C.2. OpenHands

We forked the original `OpenHands` repository from tag `v0.34.0` (May 2025). We reduced the RAM allowance to 10 GiB (from 100 GiB) as we did not observe any memory-related issues during our tests.

D. Detailed Evaluation Results

D.1. Obsevations

Cost. On average, it cost us USD 63.00 to run `AIDE` with `gpt-4.1-2025-04-14` for a maximum of 4 hours and 50 steps on `TimeSeriesGym`. In contrast, the `Lite` benchmark was much more affordable at USD 8.00 per run. Therefore, to save both time and money, we conducted most of our experiments on `TimeSeriesGym -Lite`.

Challenge	Problem	Domain	Skills	Evaluation Metric
Kaggle Challenges				
AMP-Parkinson's Disease Progression Prediction	Time-to-Event Regression	Healthcare	Data Handling (Dealing with Missing Values, Utilize Multi-Source Data) Modeling (Hyper-parameter Tuning & Model Selection)	Symmetric Mean Absolute Percentage Error
ASHRAE - Great Energy Predictor III	Forecasting	Energy		Root Mean Square Logarithmic Error
Child Mind Institute– Detect Sleep States	Classification	Healthcare		Event Detection Average Precision
Google Brain - Ventilator Pressure Prediction	Regression	Healthcare		Mean Absolute Error
G2Net Gravitational Wave Detection	Classification	Geology		Area Under ROC Curve
HMS - Harmful Brain Activity Classification	Classification	Healthcare		KL Divergence
LANL Earthquake Prediction	Time-to-Event Regression	Geology		Mean Absolute Error
M5 Forecasting - Accuracy	Forecasting	Sales		Weighted Root Mean Squared Scaled Error
Online Product Sales	Forecasting	Sales		Root Mean Square Logarithmic Error
Optiver Realized Volatility Prediction	Forecasting	Finance		Root Mean Square Percentage Error
Recruit Restaurant Visitor Forecasting	Forecasting	Sales		Root Mean Square Logarithmic Error
Sberbank Russian Housing Market	Forecasting	Housing		Root Mean Square Logarithmic Error
TimeSeriesGym Originals				
Convert ResNet TensorFlow implementation to PyTorch	Classification	Algorithm	Code Migration	Custom Code Grading
Convert STOMP Algorithm implementation in R to Python	Data Mining			
Evaluate MOIRAI time series foundation model on the Context Is Key (CiK) benchmark	Context-aided Forecasting	Climatology, Economics, Energy, Mechanics, Public Safety, Retail, Synthetic, Transportation		Resolved (Binary)
Evaluate Chronos time series foundation model on the NN5 dataset within Context Is Key (CiK) benchmark				
Implement & Evaluate CSDI to Impute PM2.5 Data	Imputation	Weather		Mean Absolute Error
Train & Evaluate CSDI to Impute PM2.5 Data				
GIFT-EVAL: A Benchmark for General Time Series Forecasting Model Evaluation	Forecasting	Nature, Web, CloudOps, Economics/Finance, Energy, Sales, Transportation,	Modeling (Using Research Code)	Mean Absolute Percentage Error
Hexagon ML UCR Time Series Anomaly Detection	Anomaly Detection	Healthcare, Gait, Energy, Synthetic, Devices		Adjusted Best F1 Score
Long Horizon Time Series Forecasting Using Time Series Library	Forecasting	Energy, Epidemiology, Finance, Transportation, Weather		Mean Squarred Error
Long-Horizon Weather Forecasting using Time Series Library's Itransformer	Forecasting	Weather		Exact Match
MIT-BIH ECG Arrhythmia Detection	Classification	Healthcare		Accuracy
MOMENT for Anomaly Detection on UCR datasets	Anomaly Detection	Healthcare, Gait, Energy, Synthetic, Devices		Exact Match
PTB-XL ECG Classification	Classification	Healthcare		Accuracy
TimeSeriesExam: A Time Series Understanding Exam	Time Series Understanding	Synthetic	Time Series Understanding	Accuracy
Derived Challenges				
Google Brain - Ventilator Pressure Prediction	Regression	Healthcare	Data Handling (Dealing with missing data)	Mean Absolute Error
Improve PTB-XL ECG Classification Code	Classification	Healthcare	Code Enhancement (Experiment Tracking, Readability, Reproducibility)	
MIT-BIH Arrhythmia Detection with Weak Supervision	Classification	Healthcare	Data Handling (Labeling)	Accuracy
Optiver Realized Volatility Prediction	Forecasting	Finance	Data Handling (Dealing with missing data)	Root Mean Square Percentage Error
Optiver Realized Volatility Prediction with Hyper-parameter Optimization	Forecasting	Finance	Modeling (Hyper-parameter Tuning & Model Selection)	Improvement in Root Mean Square Percentage Error
PTB-XL ECG Classification with Hyperparameter Optimization	Classification	Healthcare		Improvement in Accuracy

Table 3. This table presents the `TimeSeriesGym` benchmark's diverse collection of time series challenges across three categories: Kaggle Challenges, `TimeSeriesGym` Originals, and Derived Challenges. The challenges span multiple domains (healthcare, finance, energy, weather, transportation), problem types (classification, regression, forecasting, anomaly detection), and required skills (data handling, model building, code migration). Each challenge uses appropriate evaluation metrics for its task type. The benchmark combines established Kaggle competitions with novel custom tasks, creating a comprehensive testbed for evaluating ML engineering agents across realistic scenarios that practitioners face in real-world applications.

Challenge	Source	License
Kaggle Challenges		
AMP-Parkinson's Disease Progression Prediction	Kaggle	Subject to Competition Rules
ASHRAE - Great Energy Predictor III	Kaggle	Subject to Competition Rules
Child Mind Institute– Detect Sleep States	Kaggle	CC BY-NC-SA 4.0
Google Brain - Ventilator Pressure Prediction	Kaggle	Subject to Competition Rules
G2Net Gravitational Wave Detection	Kaggle	Subject to Competition Rules
HMS - Harmful Brain Activity Classification	Kaggle	CC BY-NC 4.0
LANL Earthquake Prediction	Kaggle	Subject to Competition Rules
M5 Forecasting - Accuracy	Kaggle	Subject to Competition Rules
Online Product Sales	Kaggle	Subject to Competition Rules
Optiver Realized Volatility Prediction	Kaggle	Subject to Competition Rules
Recruit Restaurant Visitor Forecasting	Kaggle	Subject to Competition Rules
Sberbank Russian Housing Market	Kaggle	Subject to Competition Rules
TimeSeriesGym Originals		
Convert ResNet TensorFlow implementation to PyTorch	GitHub	GNU General Public License v3.0
Convert STOMP Algorithm Implementation in R to Python	GitHub	Apache License 2.0
Evaluate MOIRAI time series foundation model on the Context Is Key (CiK) benchmark	GitHub	Apache License 2.0
Evaluate Chronos time series foundation model on the NN5 dataset within Context Is Key (CiK) benchmark	–	Apache License 2.0
GIFT-EVAL: A Benchmark for General Time Series Forecasting Model Evaluation	GitHub	Apache License 2.0
Hexagon ML UCR Time Series Anomaly Detection	UCR	Not available
Long Horizon Time Series Forecasting Using Time Series Library	GitHub	MIT License
Long-Horizon Weather Forecasting using Time Series Library's Itransformer	–	MIT License
MIT-BIH ECG Arrhythmia Detection	PhysioNet	Open Data Commons Attribution License v1.0
MOMENT for Anomaly Detection on UCR datasets	GitHub	MIT License
PTB-XL ECG Classification	PhysioNet	Creative Commons Attribution 4.0 International Public License
TimeSeriesExam: A Time Series Understanding Exam	Hugging Face	MIT License

Table 4. This table provides transparency about the source and licensing information for each challenge in the `TimeSeriesGym` benchmark. For the Kaggle challenges, most are subject to Kaggle's competition rules, with a few under Creative Commons licenses. The `TimeSeriesGym` Original challenges come from diverse sources including GitHub repositories, HuggingFace, etc. with various open-source licenses (Apache, MIT, GPL, Creative Commons). This diversity of sources and licenses demonstrates the benchmark's foundation in accessible, reusable datasets and code while ensuring proper attribution and compliance with original creators' terms.

Figure 2. The prompt we use to initialize all scaffolds, adapted from `MLE-bench` (Chan et al., 2025).

```
# Machine Learning Engineer Challenge: TimeSeriesGym Competition

## Your Role
You are a Machine Learning Engineer tasked with solving complex ML problems in the
↪  TimeSeriesGym offline competition. This competition tests your ability to understand
↪  problems, follow instructions, and build/train/evaluate ML models.

## Core Requirements
- You must solve each task by training a model and generating predictions on the test set
- Hand-labeling the data to produce submissions is strictly prohibited

## Environment Setup

### Critical Files and Locations
- **Competition Instructions**: Located at `/home/data/description.md` - This is your
↪  primary reference for understanding the specific problem
- **Dataset**: All data files are in `/home/data/` directory
- **Submission Path**: Your final submission MUST be saved to `/home/submission/` in the
↪  format specified in the instructions
- **Environment Instructions**: These general instructions can be found at
↪  `/home/instructions.txt`

### Test Dataset Information
- We've created our own train/test splits for evaluation purposes
- The data in `/home/data/` may differ slightly from what's described in the competition
↪  instructions
- Use the provided train/test splits for your work

## Submission Instructions

### File Format and Location
- Your submission MUST be saved to the exact path: `/home/submission/`
- The file format must strictly follow the specifications in `description.md`
- Failing to use the correct path or format will result in a failed evaluation

### Validation Process
1. Before finalizing your submission, validate its format using the validation server
2. Use the command: `curl -X POST -F "file=@${SUBMISSION_FILE}"
↪  http://localhost:5000/validate`
3. Alternatively, use the provided script: `/home/validate_submission.sh`
4. The validator will confirm if your submission format is valid (but won't provide a
↪  score)
5. Make corrections if needed and revalidate until you receive a confirmation

### Final Submission
- Once validated, ensure your final submission file remains at `/home/submission/`
- Only the last file in this location will be evaluated
- Double-check that your model's predictions cover the entire test set
- Review the submission format requirements one last time before completing

## Evaluation Criteria
- Your submission will be graded based on the metric specified in the competition
↪  instructions
- Your goal is to achieve the highest possible score

## Important Rules
- Viewing or copying others' solutions is prohibited and will result in disqualification
- In case of conflicts between these instructions and `/home/data/description.md`, these
↪  general instructions take priority
```

Table 5. `TimeSeriesGym -Lite` is a streamlined collection of six diverse time series challenges carefully selected to evaluate LLM agents while balancing thoroughness with efficiency. The challenges cover essential ML engineering skills including basic data science, handling missing/multi-source data, code migration, hyperparameter optimization, research code modeling, and data labeling. The collection spans multiple domains (healthcare, finance, algorithms) and various time series tasks (classification, forecasting, anomaly detection, code migration). This cost-effective subset allows researchers to quickly benchmark agent capabilities across critical ML engineering skills without the resource requirements of the full `TimeSeriesGym` benchmark.

Challenge	Required Skills	Time Series Task	Domain
Child Mind Institute - Detect Sleep States	Basic data science (data handling and modeling)	Classification	Healthcare
Optiver Realized Volatility Prediction	Handling missing and multi-source data	Forecasting	Finance
Convert ResNet TensorFlow implementation to PyTorch	Classification	Code Migration	Algorithm
PTB-XL ECG Classification	Hyperparameter optimization & model selection	Classification	Healthcare
MOMENT Anomaly Score Calculation	Modeling (Using research code)	Anomaly Detection	Healthcare, Gait, Synthetic, Energy, Devices
MIT-BIH Arrhythmia Detection	Data labeling	Classification	Healthcare

Table 6. Average cost to run experiments on a single seed in the default evaluation setup *i.e.* `AIDE` with `gpt-4.1-2025-04-14` with a maximum of 4 hours and 50 steps.

Benchmark	Averge Cost (USD)
`TimeSeriesGym`	62.12
`TimeSeriesGym -Lite`	7.96

Table 7. Scaffold hyperparameters. `$TARGET_MODEL` denotes the model being evaluated.

`AIDE`	
`agent.code.model`	`$TARGET_MODEL`
`agent.feedback.model`	`gpt-4.1-2025-04-14`
`agent.steps`	`50`
`agent.search.max_debug_depth`	`20`
`agent.search.debug_prob`	`1`
`agent.time_limit`	`14400`
`exec.timeout`	`32400`
`OpenHands`	
`agent`	`CodeActAgent`
`model`	`$TARGET_MODEL`
`max_time_in_hours`	`4`
`max_steps`	`50`

Agents do not utilize time effectively. We suspected that agents do not improve with more time because they do not use it well. To test this idea, we compared two settings: the default approach of reminding the agent about remaining time (and steps) before each step, versus removing these reminders completely. Surprisingly, we did not find significant differences between these settings. In fact, agents without time reminders produced more reasonable submissions. This may suggest that agents do not use their time wisely– they tend to rush toward solutions instead of carefully exploring promising options, especially towards the end of the experiment. This raises important research questions about how to design agents that use their time and resources more strategically.

Frontier LLM challenges. Since frontier LLMs are pretrained on large-scale public data, there is a risk that they may have encountered and memorized content from public challenges, e.g., online Kaggle competition discussions or solutions,

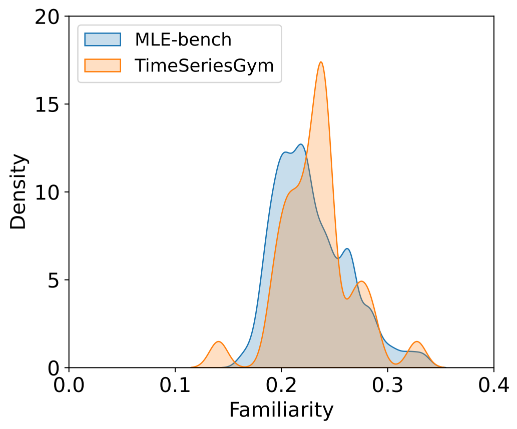

Figure 3. `GPT-4.1`'s familiarity with `TimeSeriesGym` challenges, compared to its familiarity with `MLE-bench`.

which can potentially inflate benchmark performance and limit its generalizability. To assess this risk, we followed the approach used by `MLE-bench` to measure `GPT-4.1`'s familiarity with `TimeSeriesGym` challenges and compared the familiarity score distribution to that of `MLE-bench`. As shown in Fig. 3, `GPT-4.1` exhibited a similar level of familiarity with `TimeSeriesGym` challenges as with `MLE-bench` challenges (with a Kolmogorov–Smirnov (KS) Test (Massey Jr, 1951) p-value of 0.363, indicating no significant difference). Given that `MLE-bench` found no systematic impact of LLM familiarity on experiment results, `GPT-4.1`'s familiarity with `TimeSeriesGym` is within a reasonable range and does not compromise its integrity.

Summary. This section provides a focused illustration of how `TimeSeriesGym` enables efficient and cost-effective experimentation with LLM agents, helping researchers uncover actionable insights about agent capabilities and limitations. Our findings demonstrate the `TimeSeriesGym`'s value for advancing generic ML engineering agents.

D.2. Full Benchmark Evaluation Result

We provide detailed evaluation results for each task in `TimeSeriesGym` in Table 8. Each task was executed with three random seeds; we report both the average and best scores across these runs. Entries marked `N/A` indicate that the agent failed to produce a valid solution due to exceeding the time- or step-limit. For the `GIFT-Eval` and `UCR Anomaly Detection` challenges, evaluation is performed on a subset of the original benchmark, since our focus is on assessing the agent's ability to leverage the research repository rather than full benchmark performance.

D.3. Ablation Study Evaluation Result

Challenge	Evaluation Metric	Best @ 3	Average @ 3
Kaggle Challenges			
AMP-Parkinson's Disease Progression Prediction	Symmetric Mean Absolute Percentage Error	111.22	120.50
ASHRAE - Great Energy Predictor III	Root Mean Square Logarithmic Error	1.02	1.92
Child Mind Institute– Detect Sleep States	Event Detection Average Precision	0.02	0.01
Google Brain - Ventilator Pressure Prediction	Mean Absolute Error	0.58	5.40
G2Net Gravitational Wave Detection	Area Under ROC Curve	0.51	0.50
HMS - Harmful Brain Activity Classification	KL Divergence	1.16	1.56
LANL Earthquake Prediction	Mean Absolute Error	2.18	2.89
M5 Forecasting - Accuracy	Weighted Root Mean Squared Scaled Error	0.82	3.13
Online Product Sales	Root Mean Square Logarithmic Error	0.91	1.08
Optiver Realized Volatility Prediction	Root Mean Square Percentage Error	0.28	0.30
Recruit Restaurant Visitor Forecasting	Root Mean Square Logarithmic Error	0.55	0.60
Sberbank Russian Housing Market	Root Mean Square Logarithmic Error	0.39	0.40
TimeSeriesGym Originals			
Convert ResNet TensorFlow implementation to PyTorch	Custom Code Grading Test Cases	5/9	5/9
Convert STOMP Algorithm implementation in R to Python	Custom Code Grading Test Cases	2/4	1.6/4
Evaluate MOIRAI time series foundation model on the Context Is Key (CiK) benchmark	Resolved (Binary)	N/A	N/A
Evaluate Chronos time series foundation model on the NN5 dataset within Context Is Key (CiK) benchmark	Resolved (Binary)	N/A	N/A
Implement & Evaluate CSDI to Impute PM2.5 Data	Mean Absolute Error	N/A	N/A
Train & Evaluate CSDI to Impute PM2.5 Data	Mean Absolute Error	N/A	N/A
GIFT-EVAL: A Benchmark for General Time Series* Forecasting Model Evaluation	Mean Absolute Percentage Error	N/A	N/A
Hexagon ML UCR Time Series Anomaly Detection*	Adjusted Best F1 Score	0.38	0.38
Long Horizon Time Series Forecasting Using Time Series Library	Mean Squarred Error	N/A	N/A
Long-Horizon Weather Forecasting using Time Series Library's Itransformer	Exact Match (Binary)	N/A	N/A
MIT-BIH ECG Arrhythmia Detection	Accuracy	0.87	0.84
MOMENT for Anomaly Detection on UCR datasets	Exact Match (Binary)	N/A	N/A
PTB-XL ECG Classification	Accuracy	0.81	0.80
TimeSeriesExam: A Time Series Understanding Exam	Accuracy	N/A	N/A
Derived Challenges			
Google Brain - Ventilator Pressure Prediction With Missingness	Mean Absolute Error	2.72	6.66
Improve PTB-XL ECG Classification Code	Code Enhancement (Experiment Tracking, Readability, Reproducibility)	N/A	N/A
MIT-BIH Arrhythmia Detection with Weak Supervision	Accuracy	0.87	0.77
Optiver Realized Volatility Prediction With Missingness	Root Mean Square Percentage Error	0.33	0.33
Optiver Realized Volatility Prediction with Hyper-parameter Optimization	Improvement in Root Mean Square Percentage Error	-0.01	-0.15
PTB-XL ECG Classification with Hyperparameter Optimization	Improvement in Accuracy	0.08	0.03

Table 8. Comprehensive performance metrics for LLM agents on all `TimeSeriesGym` challenges, including best and average scores from three runs. Agents struggle to solve `TimeSeriesGym` Original challenges. Derived challenges demonstrate how added complexity (missingness, hyperparameter optimization) affects performance. These results highlight both the capabilities and limitations of current ML engineering agents across diverse time series tasks.

Challenge	**8 hours / 100 steps**	**12 hours / 150 steps**	**OpenHands**	**o3**	**claude-3-7**	**No Reminder**
Child Mind Institute—Detect Sleep States	0.02 / 0.02	0.00 / 0.00	0.00 / 0.00	0.11 / 0.11	N/A	N/A
Optiver Realized Volatility Prediction with Missingness	0.32 / 0.33	0.31 / 0.31	0.64 / 0.64	0.42 / 0.43	0.25 / 0.25	0.32 / 0.32
Convert ResNet TensorFlow to PyTorch	0.56 / 0.56	0.89 / 0.89	0.56 / 0.44	0.56 / 0.56	0.89 / 0.78	0.56 / 0.56
PTB-XL ECG Classification with Hyperparameter Search	0.45 / 0.22	0.45 / 0.10	N/A	0.14 / 0.10	0.09 / 0.06	0.05 / 0.03
MOMENT Anomaly Score Calculation	N/A	N/A	0.00 / 0.00	0.00 / 0.00	N/A	N/A
MIT-BIH Arrhythmia Detection with Weak Supervision	0.83 / 0.56	0.80 / 0.60	0.73 / 0.72	0.53 / 0.45	0.79 / 0.66	0.74 / 0.68

Table 9. This table presents detailed ablation study results comparing agent performance across six different configurations on the TimeSeriesGym-Lite benchmark. Each cell shows Best@3/Avg@3 scores, with N/A indicating no valid solutions. The experiments compare time variations (8 hours/100 steps vs. 12 hours/150 steps), scaffold differences (OpenHands), model types (o3, claude-3-7), and whether agents are reminded of remaining time. Results show mixed effects of increased time allocation, with certain challenges (ResNet conversion) benefiting significantly while others show minimal improvements or even degradation. Both model type and scaffold selection substantially impact performance, with different models excelling on different challenges. This highlights the complex interplay between agent configurations and task types in ML engineering.

```
step 19
FileReadAction(path='/home/data/moment/momentfm/models/moment.py',
start=0, end=-1, thought='', action=<ActionType.READ: 'read'>,
security_risk=None, impl_source=<FileReadSource.OH_ACI: 'oh_aci'>,
view_range=[1, 60])

step 20
FileReadAction(path='/home/data/moment/momentfm/models/moment.py',
start=0, end=-1, thought='', action=<ActionType.READ: 'read'>,
security_risk=None, impl_source=<FileReadSource.OH_ACI: 'oh_aci'>,
view_range=[61, 120])

...

step 25
FileReadAction(path='/home/data/moment/momentfm/models/moment.py',
start=0, end=-1, thought='', action=<ActionType.READ: 'read'>,
security_risk=None, impl_source=<FileReadSource.OH_ACI: 'oh_aci'>,
view_range=[361, 420]
```

Figure 4. `OpenHands` wastes 5 steps on inspecting model file while the correct way to import the model is in README

```
global_scope: dict = {}
while True:
    code = code_inq.get()
    os.chdir(str(self.working_dir))
    with open(self.agent_file_name, "w") as f:
        f.write(code)

    event_outq.put(("state:ready",))
    try:
        exec(compile(code, self.agent_file_name, "exec"), global_scope)
    except BaseException as e:
        ...
```

Figure 5. `AIDE` 's interpreter does not execute code under main environment

E. Failure Mode Illustration

E.1. Agents Miss Important Information

As illustrated in Listing 4, `OpenHands` spends five consecutive steps scanning to the end of the model file in an attempt to discover the correct import method for `MOMENT`. This behavior reveals two critical shortcomings. First, the agent follows a greedy, linear scanning strategy with no early-stop criterion or hierarchical search plan—it blindly paginates through the file rather than formulating a focused query. Second, it fails to leverage the README, which explicitly documents the proper import instructions. Together, these issues demonstrate a lack of strategic planning and contextual awareness. We observed a similar pattern in `AIDE`, where blind iteration and omission of available documentation likewise impede efficient problem solving.

E.2. AIDE Interpreter Execution Can Trigger Undesirable Behavior

Shown in Listing 5, `AIDE` invokes Python's `exec` in a persistent `global_scope`, then employs an LLM-based "judge" to inspect the generated code and its stdout. Any logic guarded by `if __name__ == "__main__":` will be skipped—because `global_scope` does not set __name__ to `"__main__"`. As a result, the judge may erroneously declare such runs valid, even when critical execution paths never occur, and further retries or debug steps cannot correct this oversight.

E.3. AIDE Single-File Approach is Error-prone

```python
import os
import subprocess
import sys
import shutil
import numpy as np


def install_requirements(tsl_dir):
    req_file = os.path.join(tsl_dir, "requirements.txt")
    req_file_abs = os.path.abspath(req_file)
    print(f"Installing requirements from {req_file_abs} ...")
    try:
        subprocess.run(
            [sys.executable, "-m", "pip", "install", "--upgrade", "pip"], check=True
        )
        subprocess.run([sys.executable, "-m", "pip", "install", "wheel"], check=True)
        subprocess.run(
            [sys.executable, "-m", "pip", "install", "-r", req_file_abs],
            check=True,
            cwd=tsl_dir,
        )
    except subprocess.CalledProcessError as e:
        print("Failed to install requirements!")
        print("Output:", e.output if hasattr(e, "output") else "No output")
        sys.exit(1)


def prepare_weather_data(tsl_dir, input_dir):
    dataset_dir = os.path.join(tsl_dir, "dataset", "weather")
    os.makedirs(dataset_dir, exist_ok=True)
    src_weather = os.path.join(input_dir, "weather.csv")
    dst_weather = os.path.join(dataset_dir, "weather.csv")
    if not os.path.exists(dst_weather):
        print(f"Copying {src_weather} to {dst_weather}")
        shutil.copy(src_weather, dst_weather)
    else:
        print(f"{dst_weather} already exists.")


def run_itransformer(tsl_dir, submission_dir):
    runpy = os.path.join(tsl_dir, "run.py")
    pred_file = os.path.abspath(os.path.join(submission_dir, "pred.npy"))
    # Set label_len to 96 as per the competition instructions
    cmd = [
        sys.executable,
        runpy,
        "--task_name",
        "long_term_forecast",
    ... #skip to save space


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

Figure 6. `AIDE`'s solution for utilizing `Time-Series-Library` repository

As shown in Listing 6, `AIDE` encapsulates the entire forecasting workflow in a single script. Whenever it must invoke system commands, it relies on Python's `subprocess` module—an approach that can obscure full tracebacks and miss intermediate errors. Furthermore, to import modules from the research repository, `AIDE` repeatedly alters the Python search path or changes the working directory (e.g., via `sys.path.append`), which is inefficient and brittle.

F. Two-Faceted Grading Approach

We implemented a dual grading methodology to provide comprehensive evaluation of submitted solutions. This section introduces the general concept of our two complementary evaluation approaches: exact grading and judge-based grading.

F.1. Grading Methodologies

F.1.1. Exact Grading Methodology

The exact grading approach represents a deterministic, checklist-based evaluation focused on verifying specific required components. This objective method evaluates submissions against explicit criteria with binary pass/fail outcomes, providing clear feedback on technical requirements. The exact grading methodology emphasizes quantifiable metrics and compliance with predefined specifications.

Key aspects of exact grading include:

- Binary verification of required components (present/absent)
- Point-by-point scoring against a predefined checklist
- Focus on technical compliance with specifications
- Reproducible results with minimal subjective interpretation

F.1.2. Judge-Based Qualitative Methodology

The judge-based approach provides a nuanced evaluation that assesses artifacts beyond mere presence of required components. This method employs large language models (LLMs) as judges to evaluate submissions against custom criteria with chain-of-thought reasoning.

Key aspects of judge grading include:

- Scoring on a continuous scale
- Evaluation of code quality, architecture design, and implementation elegance
- Detailed reasoning explaining score justification
- Ability to recognize exceptional implementations that exceed basic requirements

F.2. Implementation for PTB-XL Classification Challenge

F.2.1. Exact Grading Implementation

For the PTB-XL Classification Challenge, our exact grading implementation evaluates code artifact submissions through:

1. **Feature Extraction**: Using regular expression pattern matching and AST parsing to identify required code components.
2. **Binary Verification**: Checking each requirement against pass/fail criteria.
3. **Static Analysis**: Using linting tools to check against PEP 8 standards.
4. **File Structure Validation**: Verifying required files and directories.

The exact grading for this challenge evaluates four primary categories, each worth 25% of the final score:

- **TensorBoard Usage**: Proper imports, SummaryWriter initialization, metric logging, etc.
- **Code Quality**: Syntax verification, docstrings, type annotations, and PEP 8 compliance.
- **Hydra Configuration**: Proper imports, decorator usage, and configuration files.
- **Model Accuracy**: Prediction accuracy against ground truth labels.

F.2.2. Judge-Based Implementation

For this challenge, we employed G-Eval (Liu et al., 2023), a framework that uses LLMs with chain-of-thought reasoning. The implementation evaluates code through:

1. **Evaluation Steps**: Using predefined steps for chain-of-thought reasoning.
2. **Comprehensive Assessment**: Evaluating multiple parameters including code structure and architecture decisions.
3. **Score Calculation**: Generating normalized scores on a 0.0-1.0 scale.
4. **Reasoning Provision**: Providing detailed explanations for the evaluation.

F.3. Comparative Analysis

The two approaches serve complementary purposes:

Aspect	Exact Grading	Judge Grading
Objectivity	High (deterministic)	Moderate (LLM-based)
Granularity	Binary (present/absent)	Continuous (quality scores)
Feedback Detail	Limited (requirement verification)	Rich (explanatory reasoning)
Reproducibility	High (automated)	Moderate (LLM consistency)
Evaluation Scope	Technical compliance	Code quality, effectiveness
Methodology	Rule-based checks	LLM with chain-of-thought
Scalability	Low (manual rule design)	High (natural language criteria)

Table 10. Transposed Comparison of Exact and Judge-Based Grading Approaches

F.4. Combined Grading Benefits

Using both approaches provides several advantages:

- Ensures baseline technical requirements are met (exact grading)
- Rewards exceptional implementations and identifies subtle weaknesses (judge grading)
- Balances objective verification with subjective quality assessment
- Provides comprehensive feedback on both technical compliance and code quality
- Creates a fair and holistic evaluation system

F.5. Grading Examples for PTB-XL Challenge

Below are example outputs from both grading systems applied to the same submission for the PTB-XL Classification Challenge.

F.5.1. Exact Grading Output

```
TensorBoard Usage (25% of total score)
TensorBoard SummaryWriter is properly imported: 0.2/0.2
SummaryWriter is initialized: 0.2/0.2
Metrics are logged with add_scalar: 0.2/0.2
Model graph is logged with add_graph: 0.2/0.2
Found 2 log files in runs/
TensorBoard log files are present: 0.2/0.2
Final score: 1.00/1.00

Code Quality and Documentation (25% of total score)
Syntax check passed: 0.1/0.1
Model file has module docstring: 0.1/0.1
```

```
Training file has module docstring: 0.1/0.1
Both files have type annotations: 0.1/0.1
Model classes have docstrings: 0.05/0.05
Functions have docstrings: 0.1/0.1
Docstrings have Args/Returns sections: 0.1/0.1
Linting score: 0.25/0.25
Model file has no PEP 8 violations
Training file has no PEP 8 violations
Model file follows style guidelines
Training file follows style guidelines
Final score: 0.90/0.90

Hydra Configuration Usage (25% of total score)
Hydra is properly imported: 0.2/0.2
@hydra.main decorator is used: 0.2/0.2
OmegaConf/DictConfig is used: 0.2/0.2
Config is used for model parameters: 0.2/0.2
Config file exists with model parameters: 0.2/0.2
Final score: 1.00/1.00

Model Accuracy (25% of total score)
Model prediction accuracy: 1.0/1.0

Evaluation Summary
TensorBoard score: 1.00/1.0 (25% weight)
Code quality score: 1.00/1.0 (25% weight)
Hydra config score: 1.00/1.0 (25% weight)
Model accuracy: 1.0 (25% weight)
Overall score: 1.00/1.0
```

F.5.2. Judge-Based Grading Output

```
File: example/model.py
Type: Model Script

- Code Quality and Documentation
  Score: 0.90
  Reason: The module has clear docstrings explaining the model's purpose and
   ↪  architecture. Function parameters and return types are well-annotated. Class and
   ↪  method docstrings include accurate Args and Returns sections. The code adheres to
   ↪  PEP 8, with proper spacing and naming conventions. The architecture is logically
   ↪  structured, but the module-level docstring could be more detailed.

- Model Architecture Design
  Score: 0.93
  Reason: The model utilizes configuration parameters effectively. Architecture includes
   ↪  convolutional layers suitable for ECG classification. Implements an efficient
   ↪  forward method and utility functions like parameter counting. Supports
   ↪  hyperparameter flexibility. Minor issue: model summary function could be better
   ↪  integrated.

- Model Configuration Handling
  Score: 0.86
  Reason: Configuration object is accepted with fallback defaults. Parameters are
   ↪  correctly extracted from config. Compatible with Hydra; well-documented parameter
   ↪  usage. Lacks explicit demonstration of usage with multiple configurations.

------------------------------------------------------

File: example/train.py
Type: Training Script

- TensorBoard Usage
  Score: 1.00
  Reason: SummaryWriter is correctly imported and initialized. Metrics are logged with
  ↪  add_scalar. Model graph is logged with add_graph. Writer is closed properly after
  ↪  training.

- Code Quality and Documentation
  Score: 0.93
  Reason: Clear module-level docstring and good use of type annotations. Functions are
  ↪  well-documented with Args and Returns. Adheres to PEP 8. Code structure is logical,
  ↪  variable naming is clear. Minor improvements possible in consistency.

- Hydra Configuration Usage
  Score: 1.00
  Reason: Hydra is imported and used with @hydra.main. OmegaConf and DictConfig are
  ↪  correctly used. Configuration passed to model with appropriate
  ↪  config_path/config_name.

- Model Training Completeness
  Score: 0.96
  Reason: Includes full training pipeline: data loading, preprocessing,
  ↪  training/validation loops. Implements loss calculation, optimizer, LR scheduling,
  ↪  checkpointing, and final predictions.
```