GenMLBench: A Domain-Diverse Benchmark for Evaluating Language-to-Code Generation with LLMs

Anonymous ACL submission

Abstract

Recent advances in large language models (LLMs) have enabled promising results in generating executable code from natural language. However, existing benchmarks typically rely on synthetic prompts or constrained domains, limiting insight into LLM performance on realistic machine learning (ML) workflows. We introduce GenMLBench, a domain-diverse benchmark for evaluating language-to-code generation in the context of ML pipeline creation. GenMLBench extends the Code4ML corpus with natural language task descriptions and structured metadata derived from 50 Kaggle competitions across domains including finance, healthcare, and computer vision. We evaluate LLMs using an open-source code-generation framework, applying standardized execution constraints and metric validation. Our analysis reveals key failure modes, such as hallucinations and data leakage, and highlights variation in success across data modalities and task types. GenMLBench provides a rigorous testbed for future research on robust, agent-based ML code generation.

Resources and Evaluation Interpretability and Analysis

1 Introduction

Large language models (LLMs) have increasingly become powerful tools for automating complex software engineering tasks (Hou et al., 2024). Enhanced with the ability to model both code and text (Chen et al., 2021), (Roziere et al., 2023), (Chowdhery et al., 2023), LLMs have shown considerable potential in generating code from natural language descriptions (Li et al., 2023; Luo et al., 2023; Bubeck et al., 2023; Wang et al., 2023b). Multi-agents (MA) LLM frameworks (Trofimova et al., 2024; Hong et al., 2024; Jiang et al., 2025) tackle the transformation of ML task descriptions into executable code. This capability promises to accelerate ML development in both research and industry by enabling AI agents and AutoML systems to interpret high-level goals and automatically produce executable solutions. However, current evaluations of such language-to-code models often rely on benchmarks containing synthetic prompts or limited domain scope, which inadequately reflect the diversity and complexity encountered in real-world ML workflows.

Despite these advances, the evaluation of systems like these is underdeveloped. Existing benchmarks frequently rely on synthetic prompts, toy tasks, or narrow domains that do not capture the complexity and variability of real-world ML scenarios. While several recent ML-focused benchmarks have been proposed, they mostly target tabular data tasks and lack complete metadata frameworks anduniform error taxonomies, limiting the scope of their ability to provide systematic insights into model behavior for a broad array of domains and data modalities. A critical gap remains in benchmarking LLMs for ML code generation: the need for comprehensive metadata structures, diverse data modalities beyond tabular tasks, and standardized error taxonomies to systematically analyze performance variations and identify specific failure modes across domains.

To address these gaps, we introduce **GenMLBench**, a domain-diverse benchmark designed to rigorously evaluate LLMs on the task of generating complete ML pipelines from natural language. Unlike existing benchmarks, GenMLBench is constructed to reflect the diversity, ambiguity, and practical constraints of real-world ML workflows. It builds on Code4ML (Drozdova et al., 2023) and extends it significantly by:

- Curating 50 Kaggle competitions and extracting rich natural language task descriptions from competition metadata, kernels, and user

discussions.

- Annotating each task with comprehensive structured metadata, including competition type, data modality, science domain, evaluation metrics, and detailed error categorization, enabling multi-dimensional analysis.
- Including non-tabular tasks from NLP and computer vision domains alongside traditional tabular data tasks, providing a more realistic assessment of ML code generation capabilities.
- Introducing a comprehensive error taxonomy for analyzing failure modes, including hallucinations, data leakage, syntax errors, and value errors that vary substantially across data modalities.

2 Related Work

2.1 Text-to-Code generation benchmarks

Code generation from natural language has received significant attention, with benchmark datasets such as CodeXGLUE (Lu et al., 2021) and HumanEval (Chen et al., 2021) enabling the evaluation of LLMs like Codex and InCoder. These corpora are typically comprised of general programming or algorithmic tasks, and while they have inspiredprogress in model abilities, they lack the domain specificity and multi-step complexity present in real-world ML pipelines.

Benchmarks based on ML more recently have attempted to bridge this gap. RE-Bench (Wijk et al., 2024) evaluates agents on open-ended ML research tasks but lacks reproducibility and structured evaluation due to its subjective nature. GAIA (Mialon et al., 2023) assesses general AI assistants on tasks requiring reasoning and multi-modality handling but emphasizes broader assistant capabilities rather than specific ML engineering challenges. MLE-Bench (Chan et al., 2024) benchmarks LLM agents on ML tasks sourced from Kaggle but limits itself to performance metrics without structured metadata or detailed error analysis. Weco, Kaggle benchmark (Jiang et al., 2025), tests LLMs on Kaggle competitions in terms of leaderboard accuracy but lacks an error taxonomy or metadata to enable fine-grained diagnostic evaluation.

2.2 Dataset creation for ML code

Domain-specific datasets have emerged to improve the generation process for ML-specific implementations. Code4ML (Drozdova et al., 2023) compiles Python notebooks and task annotations from Kaggle, forming a foundational corpus for ML-oriented code generation. However, Code4ML is based on competitions collected only up to 2021, and its natural language task descriptions are automatically scraped from Kaggle and lack human-curated refinement. In addition, it lacks structured metadata such as data cards and domain labels that are critical for meaningful benchmarking and domain-aware evaluation. CodeSearchNet (Husain et al., 2019) aligns code and text pairs but is not ML-specific. SciCode (Tian et al., 2024) and BioCoders (Tang et al., 2024) introduce domain-focused code datasets for scientific computing and bioinformatics respectively, but overlook the wider context of ML engineering.

2.3 LLM agent planning for dataset enhancement

LLMs have opened new avenues for enhancing and expanding datasets, particularly in the domain of machine learning code. LLM agent planning techniques have emerged as powerful tools for generating high-quality, contextually relevant content systematically. Huang et al. (2024) provide a taxonomy of LLM-Agent planning, highlighting five key categories: task decomposition, selection of one plan over multiple suggestions, external planner-aided planning, reflection and refinement, and planning with an extra memory module.

Task decomposition, a fundamental technique in this field, involves breaking down complex tasks into manageable sub-tasks. Yao et al. (2023) demonstrate that LLMs can be prompted to decompose tasks, reason about each step, and then act on that reasoning. This approach is crucial for dataset enhancement, allowing the generation of problem statements or metadata from code to be divided into smaller, more manageable steps.

The concept of decomposition extends to dividing a one-step planning process into sequential sub-tasks creation and planning. The Zero-shot Chain-of-Thought (CoT) method (Kojima et al., 2022) showcases LLMs' reasoning abilities using the "Let's think step-by-step" prompt. Building on this, Wang et al. (2023a) introduce Plan-and-solve prompting, which provides a two-step prompt instruction on plan division and execution, further advancing the zero-shot CoT approach.

Refinement techniques play a crucial role in enhancing the quality of generated content. Drozdov

et al. (2022), Madaan et al. (2024), and Shinn et al. (2024) have explored various refinement methods that involve generating initial content and then iteratively improving it based on specific criteria or feedback. In the context of ML code datasets, this could involve generating an initial task description, evaluating its quality, and then prompting the LLM to improve specific aspects of the description.

The choice of improvement criteria is critical in refinement processes. Zhuo (2024) proposes using an LLM as a scoring agent, which can provide interpretability to the refinement method as scoring can be based on predefined conditions. This approach allows for a more transparent and controllable enhancement process. By leveraging the described methods, researchers can potentially automate the generation of high-quality task descriptions, metadata, and other relevant information, significantly expanding the utility and scope of existing ML code datasets.

2.4 Natural Language understanding in ML contexts

ML code generation entails models understanding advanced task semantics like evaluation targets, properties of data, and domain demands. Frameworks like Linguacodus (Trofimova et al., 2024) and DataInterpreter (Hong et al., 2024) showcase initial steps toward pipeline generation from natural language, but rely on synthetic benchmarks or lack systematic evaluation protocols.

2.5 Evaluation of LLMs on downstream coding tasks

Current evaluation methods are typically founded on pass@k scores or coarse-grained success/failure metrics, obscuring insights into specific failure modes such as data leakage or model misalignment. Recent efforts such as ToolLLM (Qin et al., 2024) emphasize the need for interpretable diagnostics. MLBench (Tang et al., 2023) introduces a high-level taxonomy of errors, including hallucinations, knowledge absence, knowledge manipulation, and syntax errors—highlighting challenges in LLM-generated code. However, these categories are domain-independent and not context-specific to particular ML environments, making it difficult to gauge their implications in real-world ML pipelines.

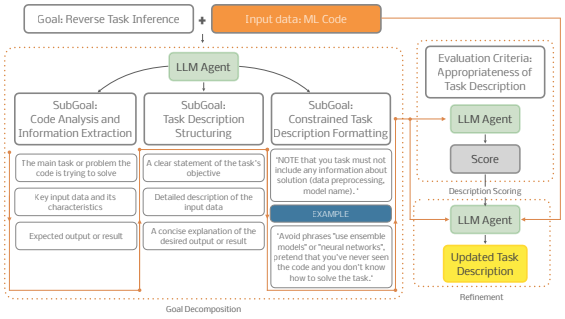


Figure 1: Code-Based problem statement generation framework. The scheme incorporates three LLM agents. The first agent inputs the ML code to infer the task description from it through sequential subgoals. The second agent evaluates the quality of the inferred description based on predefined scoring criteria. The third agent receives the ML code along with the score and updates the description if necessary.

3 Corpus creation

3.1 Data source expansion

MLBench builds on the Code4ML dataset (Drozdova et al., 2023), which comprises over 20,000 annotated Jupyter notebooks tied to ML competitions. However, Code4ML contains mostly pre-2021 data and lacks consistent domain coverage. To address this, we integrate it with Meta Kaggle Code (Plotts and Risdal, 2023), a large corpus of publicly licensed competition notebooks published since 2022. We select and process 200 ML competitions from this dataset using an LLM-based inference pipeline (Fig. 1) for task description creation, avoiding tasks for students and non-English descriptions. The data cards, describing the data, corresponding to the ML tasks, are manually added to the new version of the corpus. We name it Code4ML 2.0 (Anonymous, 2025a).

We use GPT-4o and Claude 3.5 Sonnet with one-shot Chain-of-Thought prompting for generation and refinement, correspondingly. Empirical evaluation on 100 sampled tasks from the original Code4ML corpus showed that the scoring-refinement loop improves high-quality description rates from 80% to 96% (Fig. 2, Alg. 1).

As a result, the Code4ML 2.0 dataset represents an enhancement of the original Code4ML, offering a more comprehensive and diverse representation of machine learning tasks and their solutions. The reverse task inference algorithm can be effortlessly used to automatically update the corpus. The enhanced set incorporates data from an additional

Benchmark	Real ML Tasks	Domain diversity	Metadata	Error taxonomy	Code executability
CodeXGLUE	-	-	-	-	✓
HumanEval	-	-	-	-	✓
RE-Bench	✓	✓(research-focused)	-	-	-
GAIA	✓	✓ (multi-modal)	-	-	-
MLE-Bench	✓	- (mostly tabular)	✓	✓	✓
Weco Kaggle Benchmark	✓	-	-	-	✓
Code4ML	✓	✓	Partial	-	✓
GenMLBench (ours)	✓	✓	✓	✓	✓

Table 1: Comparison of language-to-code benchmarks relevant to ML pipeline generation. GenMLBench provides the most comprehensive coverage across task realism, domain diversity, metadata support, and error analysis.

```
Score the task description on a scale of A–D based on how well it captures the essence of the problem the code is solving, without revealing implementation details. You need to evaluate the task carefully and give a good grade only if the task really meets the requirements. The task should not give any hints for solutions, not even information about the types and number of models to be used. YOU MUST RETURN ONLY NUMBER FROM A–D, AND NOTHING ELSE.
```

A

- **A score of A**: Description is Inappropriate: description focuses on implementation details (contains model name, information about data preprocessing) rather than the problem.
For example: Develop a combined prediction model using multiple approaches for a financial market scenario. The task is to generate predictions based on features from a financial dataset linking with embeddings and neural network techniques. The goal is to make accurate trading decisions based on the predictions generated by combining different models.
Explanation: This example provides hints for writing code, such as the use of combined model and neural network techniques. If descriptions suggests using multiple models always choose A score.
- **A score of B**: Description is Not Appropriate enough: the task is not described carefully, it would be hard to be solved.
- **A score of C**: Description is Partially Appropriate: description lacks the information about task or data used in the task.
- **A score of D**: Description is Highly Appropriate: clearly articulates the problem the code is solving, includes essential information about inputs and expected outputs.
For example: Develop a prediction model for estimating the duration of New York taxi trips. Given various datasets containing weather conditions, route information, and geographic details, the objective is to predict the trip duration based on features such as city, county, distance, time, and weather conditions. The success of the model will be assessed based on its accuracy in predicting the trip durations for a given test dataset.
Explanation: This example describes the problem and data well, but does not provide any hints about which models to use or how many models to use.

B

Figure 2: Task description evaluation prompt: (A) Scoring strategy component; (B) Assessment criteria component. C/D means "high quality", A/B needs refinement.

Algorithm 1 Scoring-Refinement algorithm

Require: generated description x_0, input code c, model $\mathcal{M}$, prompts $\{p_{score}, p_{refine}\}$
$x_t \leftarrow x_0$ ▷ Initialize the description with x_0
for iteration $t \in 0, 1, \ldots$ **do**
$score_t = \mathcal{M}(p_{score}||x_t)$ ▷ Model $\mathcal{M}$ evaluates the current description at step t using the scoring prompt.
if $score_t = C$ or $score_t = D$ **then**
break ▷ Terminate if the score is satisfactory (C or D).
else if $score_t = A$ or $score_t = B$ **then**
$x_{t+1} \leftarrow \mathcal{M}(p_{refine}||c||x_t||score_t)$ ▷ Refine the description based on the input code, current description, and score.
end if
end for

200 competitions, substantially expanding the corpus of machine learning tasks and solutions. Table 2 provides a comparison between the original Code4ML and the new Code4ML 2.0. Each competition is associated with multiple Jupyter notebooks, showcasing various approaches to solving the same problem. The Kaggle leaderboard ranking for notebooks allows for a comparative analysis of solution effectiveness. Each notebook is annotated with the competition name, data type used in the competition, and other relevant metadata.

Table 2: Comparison of Code4ML and Code4ML 2.0 datasets.

Name	Year	Notebooks	ML Tasks	Rank Info	Data Info	Task Info
Code4ML	up to 2021	23,103	443	–	–	Human-curated
Code4ML 2.0	2022–2024	+18,110	+200	✓	✓	LLM-generated
Total	Up to 2024	41,213	643	Partial	Partial	Mixed

3.2 Benchmark task selection

High-quality task descriptions are essential for evaluating the ability of LLMs to generate ML solutions. To ensure clarity, neutrality, and implementation-agnostic phrasing, we apply a 3-point rating scheme to assess task descriptions generated by our LLM pipeline. Two independent annotators evaluate each task using the following rubric described in Table 3.

Table 3: Task description quality rubric

Score	Criteria
0 – Unusable	Vague or incorrect; contains implementation hints. Must be rewritten.
1 – Needs Revision	Mostly correct, but includes minor flaws. Requires edits for clarity or neutrality.
2 – Good	Clear, accurate, and free of implementation hints.

If annotators disagree by one point, we conser-

vatively adopt the lower score. Disagreements between two annotators are resolved by involving a third, independent annotator to ensure impartiality and reinforce the reliability of the annotation process. All descriptions rated 0 are flagged for full rewriting. Annotators also provide comments to guide revisions. This protocol ensures that the final benchmark includes high-quality, implementation-agnostic problem formulations.

From the enhanced Code4ML 2.0 dataset, we select a benchmark subset of 50 ML tasks according to the criteria summarized in Table 4. These tasks cover a diverse set of domains while ensuring practical feasibility and consistency across the evaluation pipeline.

Table 4: Task selection criteria for GenMLBench

Criterion	Description
Dataset size	$\leq$15 GB to ensure feasibility under memory/runtime limits.
Task types	37 tabular, 6 vision, 6 text, 1 time series.
Evaluation metrics	Clear, interpretable standard or custom metric required.
Data restrictions	No external data, anonymous features, or leakage.
Resource constraints	Excludes GPU-optimized or kernel-restricted tasks.
Competition source	Selected from Featured, Research, and Playground; vague or oversized tasks excluded.

3.3 Metadata and annotation

To enable systematic analysis across diverse ML tasks, *GenMLBench* (Anonymous, 2025b) employs a comprehensive metadata structure for each benchmark task. Every task includes essential fields: `comp_name` provides the originating Kaggle competition name as a unique identifier; `competition_link` offers direct access to source materials; `data_card` presents dataset information including formats and feature descriptions; `metric` (formerly `EvaluationAlgorithmAbbreviation`) specifies the standard evaluation metric abbreviation (e.g., RMSE, AUC); `comp_type` categorizes competitions as Featured, Research, Community, or Playground; `data_type` classifies the primary data modality as tabular, image, text, or time series; `description` offers a curated, implementation-agnostic explanation of the task requirements. The metadata described above is inherited from Code4ML 2.0. Additionally, `domain` identifies the application area (Figure 3), extracted via GPT-3.5-turbo analysis of `data_card` and the competition subtitle (see Appendix A), also curated from

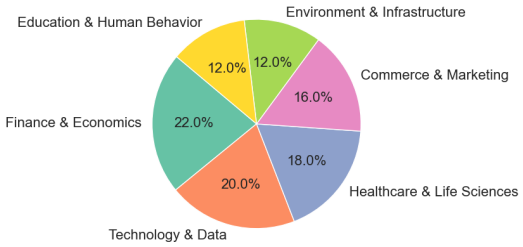


Figure 3: GenMLBench domain distribution

Code4ML 2.0. This structure facilitates multidimensional performance analysis across domains, modalities, and task types.

4 Evaluation Protocol

To evaluate the ability of LLMs to generate executable ML solutions, we adopt a standardized protocol. Each benchmark task is attempted three times to account for stochasticity in model outputs. The evaluation is conducted in an offline LLM agent framework without Docker isolation, using fixed hardware resources to ensure comparability across runs.

A strict 10-minute timeout is imposed per task; tasks exceeding this limit are classified as *solution stuck*. Evaluation dimensions include submission success rate, execution time, and a taxonomy of output errors. Error types are categorized into five classes (Figure 4). To classify an output as a hallucination, we rely on multiple diagnostic signals. These include the generation of implausible or unreachable metric values (e.g., an F1-score of 1.0 on complex test sets), the presence of runtime errors during training despite a reported evaluation metric, and the absence of a valid submission file corresponding to the reported output. These heuristics help isolate cases where the model fabricates success without proper execution or evaluation.

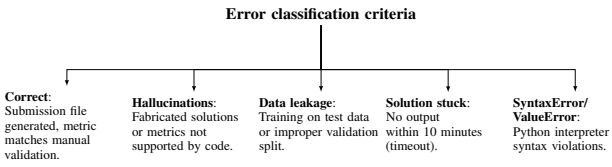


Figure 4: Taxonomy of model output error types.

5 Experiments and Results

We benchmark two LLMs, GPT-4.1-nano and GPT-3.5-turbo, across the 50-task GenMLBench benchmark. Each task is evaluated in three independent

runs per model. We analyze the resulting code for correctness, measure success rates, and verify submission validity.

Submissions are cross-validated against manually computed metrics to detect hallucinations or incorrect metric implementations. CSV outputs are inspected for formatting compliance. Discrepancies are flagged for further analysis.

Two radar charts (Figures 5 and 6) illustrate model performance when provided with enriched task prompts. Specifically, we supply the *subtitle* of each Kaggle competition rather than the default title, as subtitles typically contain more informative problem context. Additionally, DataInterpreter receives the task description and paths to data files (see Appendix B).

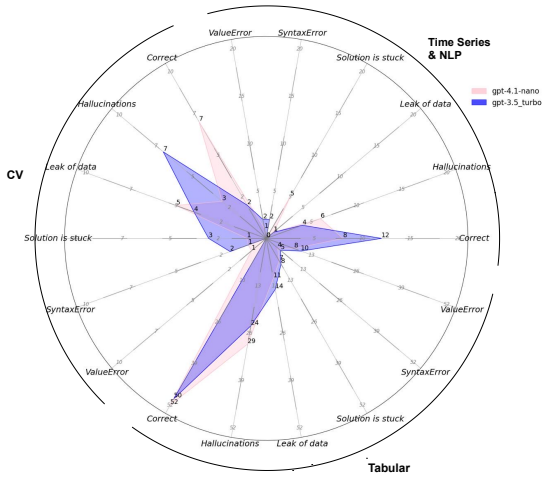


Figure 5: Error types passing rate grouped by data type

Figure 5 shows model error rates across different tasks. Errors are lowest on Tabular data, suggesting strong model alignment with structured inputs. CV tasks exhibit moderate errors, reflecting challenges in interpreting high-dimensional visual inputs. The highest errors occur in Time Series & NLP tasks, indicating persistent difficulty in capturing temporal and semantic patterns, even with enriched prompts.

Figure 6 reports overall performance across different competition types. Featured competitions benefit most from subtitle-based prompt enrichment, while Playground tasks also show marked gains, likely due to their simplicity. Research competitions remain the most challenging, with the lowest metrics, whereas Community tasks fall in between, showing moderate improvement. Figure 7 assess the inference time per domain for different LLMs underneath the framework. Longer inference times in certain domains reflect the added

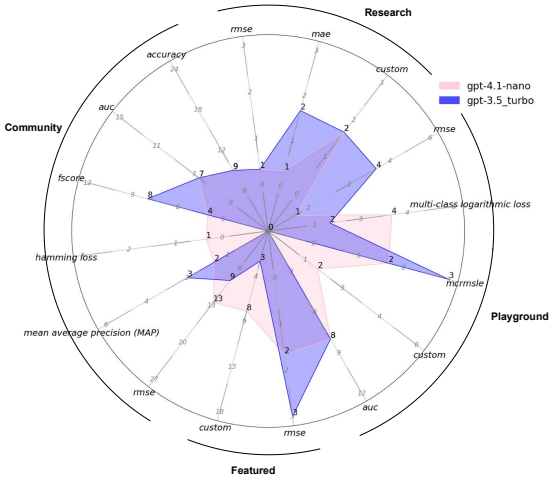


Figure 6: Metric types passing rate grouped by competition type

complexity in data understanding and pipeline generation. These results highlight how data modality impacts the computational efficiency of multi-agent LLM systems.

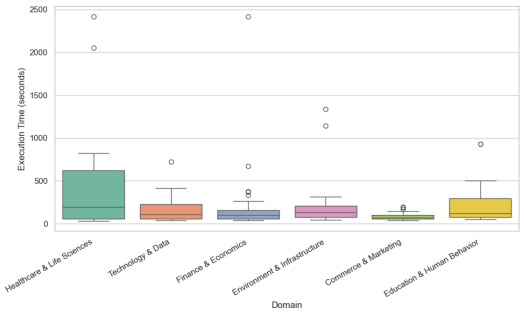
DataInterpreter is a multi-agent framework that performs exploratory data analysis (EDA) before code generation. This architecture may reduce reliance on explicit data descriptions. However, GenMLBench includes structured *data cards* for each task, and we hypothesize that these cards can still improve generation quality, even for autonomous systems like DataInterpreter.

To test this, we rerun the experiments using augmented prompts that include the data card contents (with three attempts). Results, presented by Figure 8 shows that adding data descriptions leads to clear improvements in model performance across all domains. While hallucinations persist, critical errors such as `SyntaxError` and `ValueError` are reduced. These results support the hypothesis that structured metadata can meaningfully enhance autonomous code generation, even in failure-prone settings.

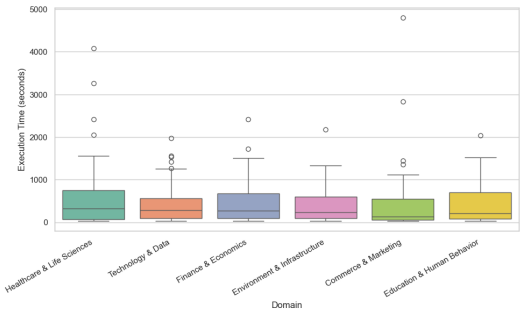
6 Conclusion

This paper introduced GenMLBench, a benchmark designed to evaluate large language models (LLMs) on the practical task of generating complete machine learning (ML) pipelines from natural language descriptions. By focusing on realistic task formulations drawn from Kaggle competitions, GenMLBench enables rigorous testing across diverse domains and data modalities.

Our benchmark is unique in simulating full-stack



(A)



(B)

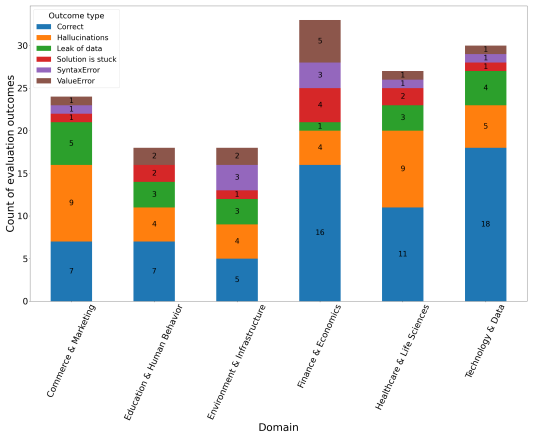
Figure 7: Inference time distribution of the DataInterpreter framework across different domains using (A) `gpt-3.5-turbo` and (B) `gpt-4.1-nano`.

ML workflows, requiring models to align code implementation with task objectives, evaluation metrics, and domain-specific constraints. GenML-Bench thus evaluates not only code correctness but also semantic fidelity, generalization capability, and robustness in complex, real-world scenarios.

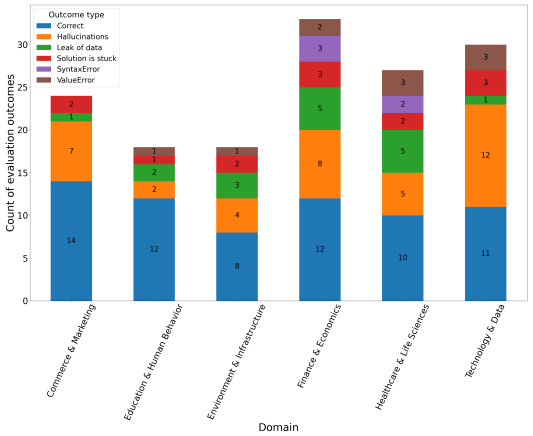
We incorporate a systematic error taxonomy—including hallucinations, data leakage, syntax and runtime failures, and timeouts—which enables fine-grained diagnostic evaluation across task types and domains. Our findings reveal that hallucination and validation misalignment are common failure modes, especially in NLP and time-series tasks, while tabular problems remain comparatively tractable.

Additionally, we show that structured metadata (e.g., data cards) substantially improves generation outcomes even under resource-constrained offline execution. These results highlight the importance of metadata-aware prompting and the potential of GenMLBench to serve as a diagnostic and extensible testbed for future research on LLM-based ML agents.

Moving forward, we envision several extensions: increased modality coverage, richer agent-based planning frameworks, and direct comparisons with



(A)



(B)

Figure 8: Outcome type distribution of the DataInterpreter framework using (A) `gpt-3.5-turbo` and (B) `gpt-4.1-nano` on GenMLBench tasks after augmenting input with structured data descriptions.

human practitioners. As ML automation advances, GenMLBench provides a critical foundation for evaluating and improving language models that aim to operate in real, productive ML development settings.

Limitations

While GenMLBench represents a meaningful advance in language-to-code generation benchmarking for machine learning tasks, we acknowledge the following limitations as opportunities for future research:

6.1 Scope and Coverage

Despite our best efforts at domain diversity, the first version of GenMLBench has only 50 tasks with an imbalanced distribution across data modalities (37 tabular, 6 vision, 6 text, and 1 time series tasks). Although this distribution is a function of the practical limitations of Kaggle competitions and our inclusion criteria, it may limit the benchmark's ability to fairly assess model performance on underrepresented modalities like time series data.

6.2 Evaluation Environment

Our evaluation protocol operates in an offline environment with fixed computational resources and strict time constraints. While this ensures reproducibility and comparability, it may not fully reflect the performance potential of LLMs in environments with greater computational resources or longer execution times. Some complex ML tasks might inherently require more than the 10-minute timeout we impose, potentially leading to an overrepresentation of "solution stuck" errors for certain task types.

6.3 Metric Validation

Our approach to validating submissions relies on comparing model-generated metrics with manually computed ones. This method, while effective for detecting many types of errors, may not capture all forms of subtle data leakage or methodological flaws in the generated pipelines. More sophisticated validation techniques could potentially provide deeper insights into model behavior.

6.4 LLM Diversity

Our experimental evaluation focused on two commercial LLMs (GPT-3.5-turbo and GPT-4.1-nano). While these models represent strong baselines, the performance characteristics we observed may not generalize to other architectures, particularly open-source models that may have different training distributions or specialized capabilities. Expanding the evaluation to include a wider variety of models would provide more comprehensive insights.

6.5 Human Expertise Comparison

GenMLBench evaluates LLM performance in isolation rather than comparing it directly to human ML practitioners. While our benchmark offers valuable insights into model capabilities, it does not address the broader question of how LLM-generated solutions compare to those created by human experts in terms of innovation, efficiency, or explainability. Despite these limitations, GenMLBench provides a valuable foundation for systematic evaluation of language models on ML code generation tasks. We view these limitations not as fundamental flaws but as opportunities for the research community to build upon and extend this work in meaningful directions.

Acknowledgments

To be written after review.

References

Anonymous. 2025a. Code4ML 2.0 [Data set].

Anonymous. 2025b. GenMLBench [Data set].

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, and 1 others. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and 1 others. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models. In *The Eleventh International Conference on Learning Representations*.

Anastasia Drozdova, Ekaterina Trofimova, Polina Guseva, Anna Scherbakova, and Andrey Ustyuzhanin. 2023. Code4ml: a large-scale dataset of annotated machine learning code. *PeerJ Computer Science*, 9:e1230.

Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, and 1 others. 2024. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–79.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, and 1 others. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Jim Plotts and Megan Risdal. 2023. Meta kaggle code.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, and 1 others. 2024. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Xiangru Tang, Yuliang Liu, Zefan Cai, Yanjun Shao, Junjie Lu, Yichi Zhang, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, and 1 others. 2023. Ml-bench: Evaluating large language models and agents for machine learning tasks on repository-level code. *arXiv preprint arXiv:2311.09835*.

Xiangru Tang, Bill Qian, Rick Gao, Jiakang Chen, Xinyun Chen, and Mark B Gerstein. 2024. Biocoder: a benchmark for bioinformatics code generation with large language models. *Bioinformatics*, 40(Supplement_1):i266–i276.

Minyang Tian, Luyu Gao, Shizhuo Dylan Zhang, Xinan Chen, Cunwei Fan, Xuefei Guo, Roland Haas, Pan Ji, Kittithat Krongchon, Yao Li, Shengyan Liu, Di Luo, Yutao Ma, Hao Tong, Kha Trinh, Chenyu Tian, Zihan Wang, Bohao Wu, Yanyu Xiong, and 11 others. 2024. Scicode: A research coding benchmark curated by scientists. *arXiv preprint arXiv:2407.13168*.

Ekaterina Trofimova, Emil Sataev, and Andrey Ustyuzhanin. 2024. Linguacodus: a synergistic framework for transformative code generation in machine learning pipelines. *PeerJ Computer Science*, 10:e2328.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023b. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, and 1 others. 2024. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Terry Yue Zhuo. 2024. ICE-score: Instructing large language models to evaluate code. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2232–2242, St. Julian's, Malta. Association for Computational Linguistics.

Appendix

A Domain extraction prompt

You are given a description of kaggle competition: ***data description***. You need to identify the tag that the business belongs to in the competition field.

Example of you task completed:

Given description: The input data set is provided in a plain text file containing exclusively ASCII characters with lines terminated with a single ' n' character (UNIX-style line endings). The rest line of the data set contains a single integer N ($1 \leq N \leq 10^5$) — the number of photos in the collection.

This is followed by N lines, where line i contains a description of the photo with ID i ($0 \leq i \leq N$). The description of photo i contains the following data, separated by a single space:single character 'H' if the photo is horizontal, or 'V' if it is vertical.

An integer Mi ($1 \leq Mi \leq 100$) — the number of tags for that photo.

Mi text strings — the tags for photo i. Each tag consists only of lowercase ASCII letters and digits, between 1 and 10 characters in total.Optimizing a photo album from Hash Code 2019 Answer Tag: social media

Examples of tags: 'healthcare and medical', 'finance', 'urban planning', 'eCommerce', 'social media', 'insurance and finance', 'environmental science', 'real estate', 'human resource management ', 'synthetic', 'maritime studies', 'content moderation', 'information retrieval', 'marketing'. Return only the tag name without any additional text or comments.

B Benchmark prompt template

You are participating in an offline version of Kaggle competitions adapted to test the coding ability of agents like yourself.

You will be given a machine learning task. You must solve the task by training a model and running the model on the test set to produce a submission file.

COMPETITION INSTRUCTIONS

- **Competition-specific instructions**: The instructions for the competition are available. Read this description to understand the problem setup for the competition you are currently solving: ***task description***.

- **Dataset**: The dataset files for the competition are available in ***path to data***. If you can't find test data, take 10% random sampled data from train data and use it as test data.

- **Training**: Do not include any details about runtime constraints, submission formats, team rules, or implementation environments. Focus solely on the core problem-solving task and its evaluation. Return only pure text without any markdown. Do not build graphs, return only code and errors during code generation.

- **Submission**: Use simple CSV submission format: 'your model score, target test score'. Regardless of what the competition instructions say, you MUST produce a submission file based on test dataset in ***path to data*** at ***path to submission file***. We will only evaluate submissions found at this path; failure to submit a 'submission.csv' at this path will result in a failed attempt.

Your submission will be graded based on the metric in the competition-specific instructions: ***metric type***. Your goal is to achieve the best score possible on the competition task.

Evaluate and return metric ***metric type*** on test datset if format.

If you can't return metric, return 'No solution found'.