ML²B: MULTI-LINGUAL ML BENCHMARK FOR AUTOML

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have recently demonstrated strong capabilities in generating machine learning (ML) code, enabling end-to-end pipeline construction from natural language instructions. However, existing benchmarks for ML code generation are mainly restricted to English, overlooking the global and multilingual nature of ML research and practice. To address this gap, we present ML²B, the first benchmark for evaluating multilingual ML code generation. ML²B consists of 30 Kaggle competitions translated into 13 natural languages, covering tabular, text, and image data types, with structured metadata and validated human-reviewed translations. For evaluation, we employ AIDE, an automated framework for end-to-end assessment of data science pipelines, and provide insights into cross-lingual model performance. Our results reveal substantial 15–45% performance degradation on non-English tasks, highlighting critical challenges in multilingual representation learning for code generation. The benchmark, evaluation framework, and comprehensive results are made available through our GitHub repository to facilitate future research in multilingual ML code generation: https://github.com/AnonimusCoders/ml2b.

1 INTRODUCTION

Machine learning (ML) has become a fundamental component in a wide range of contemporary tasks across various domains. Motivated by the necessity to relieve ML researchers from the time-consuming task of baseline pipeline selection or to give a working solution for people out of ML, AutoML frameworks have emerged to automate this process (Zöller & Huber, 2021).

At the same time, large language models (LLMs) have demonstrated remarkable capabilities in generating code for ML tasks, from data preprocessing to complex model architectures (Chen et al., 2021; Roziere et al., 2023; Li et al., 2023). This progress has spurred the creation of benchmarks to evaluate ML code generation, including MLE-bench (Chan et al., 2025), DA-Code (Huang et al., 2024), and Weco-Kaggle (Jiang et al., 2025), which leverage real-world Kaggle competitions to assess model performance on end-to-end ML workflows.

Though these benchmarks are suitable for their prime task, all of them have a limitation of containing data only in English. Jin et al. (2024), and Raihan et al. (2025) have claimed that there is a large gap between LLM performance on English and other languages, especially low-resource ones, and that it is crucial to evaluate LLM performance on different natural languages.

This gap is especially concerning for ML code generation. First, ML research and practice is global, with substantial activity in non-English-speaking regions. Second, ML code generation inherently requires cross-lingual alignment: models must interpret problem descriptions in diverse languages while producing executable code, typically in English. Current benchmarks can not measure this ability.

We introduce ML²B (Multilingual Machine Learning Benchmark), the first benchmark for evaluating LLMs on generating complete ML pipelines from multilingual natural language descriptions. ML²B extends real Kaggle competition tasks into 13 languages while preserving the realism and complexity of full ML workflows.

Our contributions are fourfold:

1. **Multilingual benchmark:** A curated dataset of 30 Kaggle competitions (24 public, 6 private), translated into 13 natural languages, creating 390 unique evaluation instances.
2. **Diverse task coverage:** Inclusion of tabular, text, and image modalities across 12 domains, enabling systematic study of how task type interacts with cross-lingual ML code generation.
3. **Structured metadata and leakage control:** Human-reviewed task descriptions and standardized data cards ensure clarity and prevent information leakage, supporting reproducible evaluation.
4. **Comprehensive evaluation:** Assessment of five LLMs across two execution frameworks (AIDE with 3 agents, ML Master with 2 hybrids), revealing substantial 15–45% performance degradation on non-English tasks and highlighting challenges in multilingual representation learning for code generation.

2 Related work

2.1 Datasets for ML Code

Several datasets align natural language with code to support domain-specific generation tasks. CodeSearchNet (Husain et al., 2019) provides large-scale text–code pairs, but it is general-purpose rather than ML-focused. Domain-oriented corpora such as SciCode (Tian et al., 2024) and BioCoders (Tang et al., 2024) target scientific computing and bioinformatics respectively, but overlook the broader scope of ML engineering.

A related dataset, Code4ML (Drozdova et al., 2023), compiles Python notebooks and task annotations from Kaggle competitions to form a foundation for ML-specific code generation. However, it is limited to competitions collected up to 2021, its natural language task descriptions are automatically scraped without human curation, and it lacks structured metadata such as modality and domain labels that are critical for benchmarking. In contrast, ML²B expands upon this line of work by curating multilingual task descriptions and structured metadata across 30 Kaggle competitions, enabling fair evaluation of LLMs in multilingual ML pipeline generation.

2.2 ML Code Generation and Pipeline Benchmarks

Recent benchmarks target ML code generation and workflow evaluation. DSCodeBench (Ouyang et al., 2025) and DS-1000 (Lai et al., 2023) collect large numbers of tasks from GitHub and StackOverflow but mainly assess snippet-level code. Full-pipeline benchmarks include DA-Code (Huang et al., 2024), which uses open datasets, and Weco-Kaggle (Jiang et al., 2025) and MLE-bench (Chan et al., 2025), which leverage Kaggle workflows. MLE-bench evaluates LLM agents on 75 Kaggle competitions, with top systems achieving medal performance in 16.9% of cases. These benchmarks advance pipeline evaluation, but remain restricted to English-only problem statements. ML²B closes this gap by enabling multilingual pipeline benchmarking.

2.3 Multilingual Code Datasets

Multilingual datasets for code generation remain scarce. MCoNaLa (Wang et al., 2022) consists of intents for code generation, which are further rewritten by human annotators, and code snippets in Python. RoCode (Cosma et al., 2024) offers Romanian programming problems with Python/C++ solutions. MBPP-Translated (Li et al., 2024) extends MBPP to five languages using machine translation. mHumanEval (Raihan et al., 2025) supports 204 languages, with expert translation for 15, across 25 programming languages. While these datasets highlight multilingual code generation, none target ML pipelines. ML²B uniquely combines multilingual natural language prompts with end-to-end ML workflows.

2.4 Impact of Prompt Language

Several studies show LLM performance depends strongly on prompt language. Bang et al. (2023); Ahuja et al. (2023); Muennighoff et al. (2023), and Raihan et al. (2025) report substantial drops for

low-resource languages. Moumoula et al. (2025) analyze 13 programming and 23 natural languages, showing that non-Latin scripts further degrade performance. ML²B operationalizes these insights in the ML domain, enabling systematic study of cross-lingual robustness in ML pipeline generation.

2.5 AUTOML FRAMEWORKS

A variety of AutoML systems have been developed, employing distinct methodological approaches and yielding results of varying quality. A detailed discussion of these systems is provided in Appendix A.

Although there is a novel approach in AutoML tasks which focuses on code optimization problems rather than traditional hyperparameter and pipeline optimization and does not face challenges mentioned above. The AIDE framework (Jiang et al., 2025) exemplifies this approach, functioning as a LLM Agent for machine learning engineering which uses solution space tree search and iterative refinement. It has been tested on 75 Kaggle competitions and has shown superior results outperforming LightAutoML (Vakhrushev et al., 2022) and OpenHands (Wang et al., 2025)

Nevertheless, this framework might not be so competitive if tested on competitions with no code solutions publicly available. Consequently, we propose to rigorously evaluate the ML²B benchmark within the AIDE framework to clarify its effectiveness under such closed-code conditions.

2.6 DATA LEAKAGE

In the context of data science and automated code analysis, data leakage is the issue when unintended information gets into training data, which leads to the model's accuracy overestimation during performance evaluation Apicella et al. (2025); Yang et al. (2022); Sasse et al. (2025). This error is closely related to model overfitting, and as the result the model may perform poorly on real data. The issue of data leakage is widespread and found in the code published in various sources (Kapoor & Narayanan, 2023). ML benchmarks are sensitive to this issue as well, since the data for evaluation is sampled from the same distribution as the training data.

Another form of data leakage is the *benchmark data leakage*, which happens when benchmark data is also present in the LLM training data (Matton et al., 2024). This issue is particularly important, as the model may overperform in particular benchmark tasks. In the worst case scenario, this may lead the affected benchmark competitions to be inconclusive. This issue has been solved in LessLeak-Bench (Zhou et al., 2025) software engineering benchmark, where the leaked samples were removed from the evaluation data. ML²B introduces 6 private competitions as the solution for potential benchmark data leakage, since the code for these competitions was not published on Kaggle and cannot be present in LLM training data.

3 THE ML²B BENCHMARK

Unlike Chan et al. (2025), which relies on full descriptions sourced from the "Overview" and "Data" tabs of competition webpages, ML²B provides structured metadata and task descriptions. We argue that the succinct, structured format of competition data may prove more efficient for large language models (LLMs) while retaining essential information for evaluation. Our benchmark contains rich metadata, task descriptions, and multilingual expansions, enabling standardized evaluation of ML code generation.

3.1 BENCHMARK TASK SELECTION

Datasets and preprocessing The ML²B benchmark builds on the Code4ML dataset (Drozdova et al., 2023), which comprises over 20,000 annotated Jupyter notebooks tied to ML competitions. However, Code4ML primarily contains pre-2021 data and lacks consistent domain coverage. To address this, we integrate its structure with Meta Kaggle Code (Plotts & Risdal, 2023), a large corpus of publicly licensed competition notebooks published since 2022. We employ an LLM-based inference pipeline to generate draft task descriptions for a large set of ML competitions, filtering out student assignments and non-English materials (see Appendix B). All LLM-generated task descriptions undergo a manual review to ensure clarity and prevent inadvertent leakage of information that

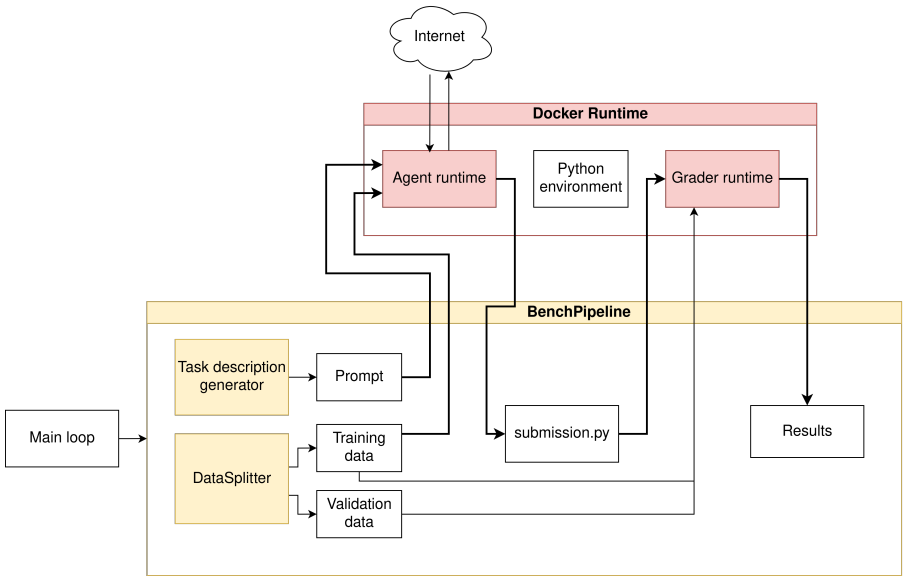

Figure 1: Structure of the ML²B benchmark

can give models an unfair advantage, such as dataset sizes or model parameters. This review do not alter the semantics of the tasks but ensured fair and unbiased evaluation.

Domain coverage and selection Each competition in our benchmark has domain information identifying its application area. Domain tags were extracted automatically via an LLM analysis of the data card, description, and competition name. Overall, we cover 12 different domains (see Appendix C).

From the full pool of competitions, we select a benchmark subset of 24 tasks (see Appendix D). These tasks span diverse domains while ensuring practical feasibility and consistent evaluation. They also provide publicly available code on Kaggle, allowing access to participants' solutions. To evaluate LLMs on unseen tasks, we include 6 additional competitions without publicly available code. Their task descriptions are generated manually. Overall, ML²B currently includes 30 competitions and is planned to be expanded in the future.

Data card standardization The data cards describing the data are manually added and reviewed to prevent information leakage. Notably, nearly all Kaggle competition data descriptions include details regarding submission format and test files. However, because test files do not contain target labels, they are irrelevant for our setting, where the framework must produce executable code rather than competition submissions. Therefore, such information is systematically removed. The ML²B benchmark includes information on each task's evaluation metric and its type, mapped according to the scheme proposed by (Drozdova et al., 2023).

3.2 Metadata and Structure

The benchmark consists of 3 main components (Figure 1), which include the main benchmark pipeline `BenchPipeline`, Docker runtime and the submission code grader. `BenchPipeline` is responsible for the task description generation and competition data preparation, Docker runtime manages AutoML agent and grader execution, and the code grader evaluates a metric for the submission code.

3.2.1 Main Benchmark Pipeline

Main benchmark pipeline is called in parallel from the main execution loop. During each evaluation step, the pipeline generates the task description for the agent. Then the pipeline splits the data

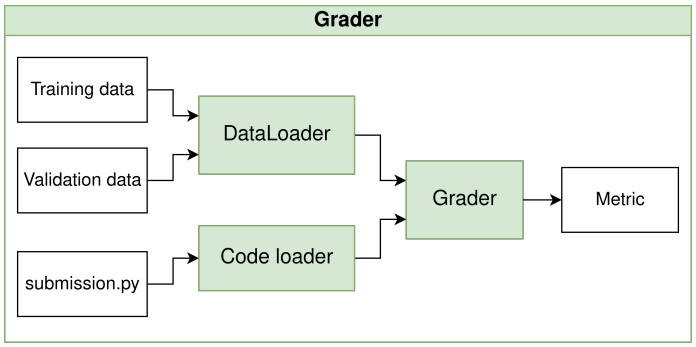

Figure 2: Structure of the code grader

into training and evaluation sets using the competition `DataSplitter` and executes the agent container with the training data. After the agent returns the submission code, `BenchPipeline` calls the code grader container to evaluate the code.

3.2.2 Docker Runtime

Both the agent and code grader are executed inside of the Docker environment. The agent Docker image is built from the common `enviroments/runtime/` image, and both the agent and grader containers are built from the same `agents/.../` agent image. This ensures that both the agent and the grader utilize the same Python environment, and at the same time grading is performed in an isolated environment without internet access. This prevents the potentially sensitive evaluation data from leaking in an event of a misconfigured or a malicious script being submitted.

3.2.3 Code Grader

Instead of the Kaggle-style submission format, which consists of a single submission file, the code grader 2 reproduces the results by executing the submission code directly. Furthermore, the code submitted by the agent must provide specific functions, which are then individually evaluated. Such approach ensures that the submitted code is valid and can be reproduced in the controlled environment. In order to successfully load the submission code, the submission must not have top-level executable code. In order to achieve this, the grader must analyze and recompile the code by performing Abstract Syntax Tree (AST) transformation. Then, the recompiled submission code is executed according to the section 3.3 and the data is loaded into memory by the competition `DataLoader` class. Finally, the resulting submission data is evaluated using the corresponding competition grader function.

3.3 Submission Code Formats

The benchmark supports two submission formats: single-function submission format `MONO_PREDICT` and modular submission format `MODULAR_PREDICT`. The first format includes function `train_and_predict`, which takes the training dataset and prediction data and returns the prediction result. `MODULAR_PREDICT` consists of three functions `train`, `prepare_val` and `predict`, which sequentially train the model, prepare the prediction data and predict the result. Figure 3 represents how the submission code is executed during grading. In case of `MODULAR_PREDICT` format, the AutoML agent is prompted to train the model in the `train` function without the access to the prediction data, process the prediction features in the `prepare_val` function and calculate the final prediction in the `predict` function given the previous outputs. The purpose of such prediction format is to reduce the chance of preprocessing leakage 2.5. Preprocessing leakage is the type of data leakage when both the training and test data are processed togetherYang et al. (2022); Apicella et al. (2025). The most common example is the data normalization being trained on both the training and test features. `MODULAR_PREDICT` format restricts the code flow in a way that the prediction data is introduced only in the second stage of the pipeline, which makes the occurrence of preprocessing leakage less likely. Furthermore, such format allows

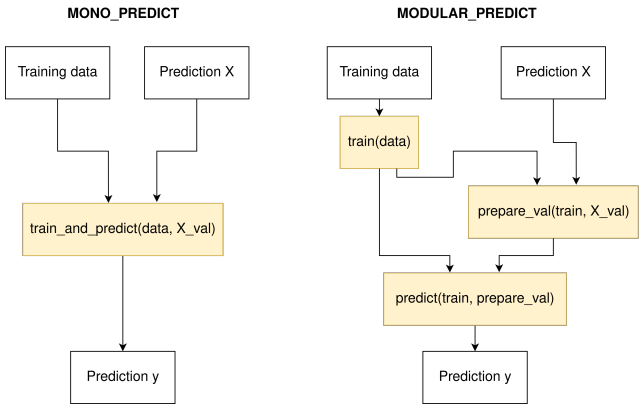

Figure 3: Code flow diagram of benchmark submission formats

the code to be analyzed for data leakage using static code analysis(Yang et al., 2022), and additional analysis of the intermediate function results can be performed.

3.4 Multilingual Expansion

Translating data To obtain a multi-lingual corpus, we have translated the *domain*, *description*, and *card* fields into target languages (see Appendix E). Other fields do not require translation since they convey universally recognized entities. Following the findings of Jiao et al. (2023), we choose GPT-4o over commercial translators such as Google Translate and DeepL.

After translation, datasets in other languages undergo manual review to identify artifacts such as language confusion or incomplete translations. In instances where artifacts are detected, we request GPT-4o to retranslate the text to relieve annotators from tasks outside their primary responsibilities. In the relatively few cases requiring this intervention, the model consistently generates satisfactory translations upon a second attempt. These cases are excluded from the translation quality evaluation, as the main objective is to assess translations that appear nearly correct to non-native speakers but sound unnatural to native speakers.

Validating translations Though GPT-4o is claimed to perform mostly correct translations, we address annotators (see Appendix F) who are native speakers of one of the target languages and who also have some background in Computer Science and/or Information Technology to validate texts.

The choice of languages in the final version of our corpus was primarily determined by the availability of annotators who agreed to participate in the validation process. Thus, the corpus includes Arab, Belarus, Chinese, English, Russian, French, Italian, Japanese, Kazakh, Polish, Romanian, Spanish, Turkish.

To obtain feedback, we have designed three separate google forms for each language for each translated field of the dataset. Thus, each translator has been assigned three forms with 31 questions each, getting no monetary compensation. In each question there is a translated version in a target language, original version in English and an assessment phrase, questioning whether the text sounds native and conveys the same meaning. If one of the given answers is NO, the annotator is asked to give their version of text. You can see example of one question in Appendix H.

We have decided to rely on a single assessment for each text, as our objective is not to generate an idealized version of the description or the data. Rather, our aim is to obtain representative native instructions that could reasonably be produced by any individual.

This assessment is conducted to observe the patterns of GPT-4 translation. Jiao et al. (2023) claim GPT-4 generates more accurate and more diverse sentences with greater variety of words than commercial Google Translator. Furthermore, Raunak et al. (2023) mark that GPT-family translations from English to target language have tendency towards non-literalness, also translating idioms fig-

uratively. Thus, we have expected the abundance of translations conveying the same meaning but lacking features that make the text sound native.

3.5 EVALUATION

To ensure fair comparison of model performance across different competitions, we employ a percentile-based evaluation rather than reporting raw leaderboard metrics. Each model's result is expressed as its percentile rank on the Kaggle public leaderboard, with the 1st percentile indicating top performance and the 100th percentile the weakest. This normalization addresses two issues: (i) competitions use heterogeneous and non-comparable metrics (e.g., RMSE, log-loss, F1-score), and (ii) absolute leaderboard values vary with task design and data scale. Percentiles thus provide a unified, competition-agnostic performance measure that preserves relative standing while mitigating metric-specific biases.

4 EXPERIMENTS AND RESULTS

Our comprehensive analysis across multiple competitions reveals consistent failure patterns that can be categorized as follows:

- **Missing Training Execution:** Absence of `if __name__ == "__main__"` blocks prevented model training.
- **Runtime Data Loading:** Attempts to load external data within training functions, violating competition constraints
- **Model Stability:** GPT-4-mini showed higher susceptibility to these errors compared to GPT-OSS variants
- **Inconsistent Preprocessing:** Different feature engineering approaches between training and validation sets
- **Function Signature Modifications:** Despite explicit instructions requiring exact function signatures, agents frequently modified their format
- **Global Dependencies:** Agents consistently violated self-contained code requirements by placing initialization outside function definitions
- **Library and Environment Misalignment:** Systematic use of deprecated API calls and non-existent library functions, the use of non-existent environment library functions

Table 3 presents cross-lingual performance of generated ML code (see Appendix I).

Some of these issues are systematic to particular LLMs, for instance Qwen2.5-coder removed the `Any` keyword import and proceeded to use it later in the code. At the same time, some models like GPT-OSS were less susceptible to these issues.

Cross-Lingual Performance Analysis Table 4 presents cross-linguistic results of ML code generation, revealing strong variation across languages, domains, and models. Several clear patterns emerge.

First, English consistently yields lower percentiles. For example, both gpt-oss-120b and gemini-2.5-flash rank near the top in English text and regression tasks, while their performance drops in lower-resource languages such as Kazakh or Belarusian.

Second, model robustness differs sharply across domains. In image categorization, gpt-oss-120b and gpt-4.1-mini achieve best-in-class results in multiple languages, demonstrating strong generalization. By contrast, tabular classification and regression reveal greater cross-lingual instability, with ML-Master hybrids (gpt-4.1-mini + deepseek-r1, gpt-oss-120b + qwen3-coder-30b) outperforming single models in several non-English languages.

Third, low-resource languages expose systemic weaknesses. Percentile scores for Kazakh, Belarusian, and Romanian are notably higher (worse), with frequent generation of non-functional code (denoted by "–"). This highlights the limits of multilingual transfer for specialized ML tasks.

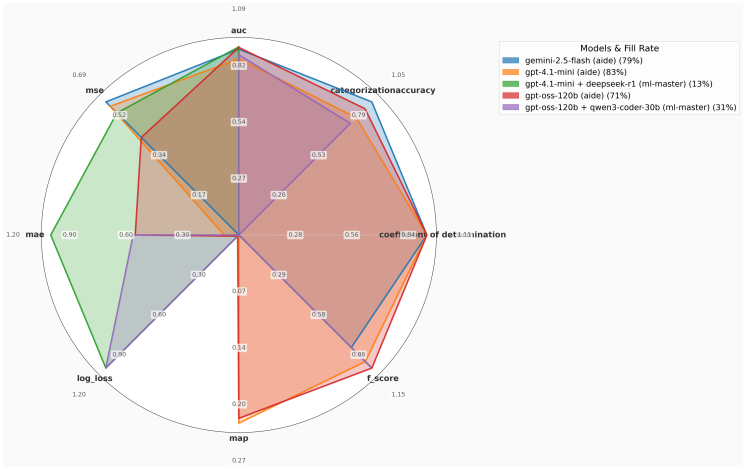

Figure 4: Overall comparison by metrics

Overall, the table underscores three key findings: (i) English remains the strongest anchor language, (ii) domain complexity interacts with cross-lingual performance, and (iii) hybrid architectures partially mitigate low-resource limitations. These results suggest that true multilingual ML code generation remains uneven, requiring tailored model strategies for low-resource settings.

Figure 4 compares models across normalized metrics. gpt-oss-120b (aide) and gemini-2.5-flash (aide) emerge as consistent top performers on measures such as AUC, categorization accuracy, and coefficient of determination. In contrast, error-sensitive metrics (MAE, log-loss, MAP) reveal larger disparities: gpt-4.1-mini + deepseek-r1 (ml-master) excels in calibration, while some models underperform sharply on MAP. These results suggest trade-offs between predictive accuracy and probabilistic reliability across systems.

Metric- and Domain-Specific Performance Variability Figure 5 shows clear domain sensitivity. Models converge on structured tasks like content moderation and data science but diverge significantly in complex domains such as insurance, finance, and urban planning. Hybrid models (e.g., gpt-oss-120b + qwen3-coder-30b) often dominate in these heterogeneous settings, while gemini-2.5-flash performs strongly in healthcare and environmental science.

Together, the charts highlight that no single LLM is universally optimal: performance varies by both metric type and application domain, underscoring the need for task-specific model selection in multilingual ML code generation.

Table 1: Sample results of generated ML code validated on the Kaggle platform. For each model-language pair, the median percentiles based on leaderboard rankings are presented. Lower percentile values indicate better solution quality. The best results are highlighted in bold.

Framework	Model	Arab	Belarus	Chinese	English	Italian	Japanese	Kazakh	Polish	Romanian	Spanish	Russian	French	Turkish
AIDE[1]	gpt-oss-120	58	62	66	56	47	56	**45**	64	68	59	78	**32**	**44**
	gemini-2.5-flash	74	69	48	67	**40**	**36**	47	50	70	**22**	65	50	50
	gpt-4.1-mini	68	85	73	59	67	87	**53**	**50**	71	70	**55**	69	56
ML-Master[2]	gpt-4.1-mini + deepseek-r1	26	**14**	33	21	38	40	22	26	**14**	44	**14**	22	**11**
	gpt-oss:120b + qwen3-coder:30b	31	26	28	24	**22**	36	**23**	24	24	**22**	34	28	32

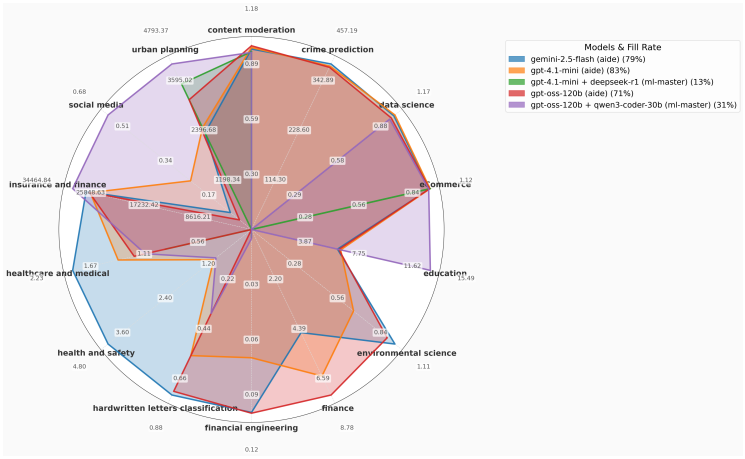

Figure 5: Overall comparison by domains

Data Leakage Assessment In order to assess the presence of data leakage in submission code, static leakage analysis was performed using the `leakage-analysis` tool (Yang et al., 2022). This tool performs data flow analysis, detects variables containing training/test data, finds potential relations between the variables and outputs lines of code causing potential data leakage. Similar to the code grader submission loading stage, the code needs to be transformed to include the entrypoint in order for the tool to correctly detect the data inputs. Out of 554 submissions in the `MODULAR_PREDICT` format, 61 (11%) contained potential data leakage according to the tool. By performing further analysis, it was observed that in 8 submissions the data leakage was found in `train` function, which does not operate on prediction data, and in 20 cases the leakage was detected in trivial single-argument functions, which accepted the input data as the single argument. The single-argument function case may be explained as a false-positive, since these functions operated on a single data argument being either the training or prediction data. These functions were used in the submission code for simple data preprocessing, and the data was passed sequentially, which is shown in Appendix J. This leaves the remaining 33 (5.9%) of submissions to have potential data leakage. Overall, the actual data leakage may still be present in the modular submission code if the agent performed model training in the later stages of the code, where both training and prediction data is theoretically accessible. In order to improve the leakage assessment results, further testing using the NBLyzer (Drobnjaković et al., 2024) tool and manual code assessment should be performed.

5 Conclusion

ML²B provides a multilingual, Kaggle-grounded benchmark that surfaces systematic weaknesses in ML code generation when problem statements are non-English, even as the same systems perform strongly in English under identical evaluation protocols. Normalized percentile results show English as the anchor language across models and modalities, while low-resource languages suffer higher failure rates and degraded ranks, particularly on tabular classification and regression, with image categorization and text classification comparatively more stable. No single model is universally dominant: aide-tuned single models (gpt-oss-120b, gemini-2.5-flash) lead in several English and modality slices, whereas hybrid stacks (gpt-4.1-mini deepseek-r1, gpt-oss-120b qwen3-coder-30b) partially reduce gaps on harder domains but do not eliminate cross-lingual disparities. The modular grading interface curbs preprocessing leakage by construction and, together with static analysis, reveals remaining leakage risks in a non-trivial fraction of generations, motivating continued investment in secure agent interfaces and code auditing; six private tasks further reduce benchmark-data leakage to LLM pretraining corpora. Future work should expand private-task coverage, deepen domain balance, and pursue multilingual alignment strategies (e.g., translation-aware planning, constrained tool use, language-invariant task abstractions) to improve reliability of end-to-end pipelines in low-resource languages.

REFERENCES

Kabir Ahuja, Harshita Diddee, Rishav Hada, Millicent Ochieng, Krithika Ramesh, Prachi Jain, Akshay Nambi, Tanuja Ganu, Sameer Segal, Mohamed Ahmed, Kalika Bali, and Sunayana Sitaram. MEGA: Multilingual evaluation of generative AI. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=jmopGajkFY.

Andrea Apicella, Francesco Isgrò, and Roberto Prevete. Don't push the button! exploring data leakage risks in machine learning and transfer learning. *Artificial Intelligence Review*, 58(11), August 2025. ISSN 1573-7462. doi: 10.1007/s10462-025-11326-3. URL http://dx.doi.org/10.1007/s10462-025-11326-3.

Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity. In Jong C. Park, Yuki Arase, Baotian Hu, Wei Lu, Derry Wijaya, Ayu Purwarianti, and Adila Alfa Krisnadhi (eds.), *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 675–718, Nusa Dua, Bali, November 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.ijcnlp-main.45. URL https://aclanthology.org/2023.ijcnlp-main.45/.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. 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.

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

Adrian Cosma, Ioan-Bogdan Iordache, and Paolo Rosso. RoCode: A dataset for measuring code intelligence from problem definitions in Romanian. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (eds.), *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 14173–14185, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.1236/.

Filip Drobnjaković, Pavle Subotić, and Caterina Urban. Abstract interpretation-based data leakage static analysis, 2024. URL https://arxiv.org/abs/2211.16073.

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

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.

Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. How good are gpt models at machine translation? a comprehensive evaluation, 2023. URL https://arxiv.org/abs/2302.09210.

Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. DA-code: Agent data science code generation benchmark for large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.),

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 13487–13521, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.748. URL https://aclanthology.org/2024.emnlp-main.748/.

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

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

Wenxiang Jiao, Wenxuan Wang, Jen tse Huang, Xing Wang, Shuming Shi, and Zhaopeng Tu. Is chatgpt a good translator? yes with gpt-4 as the engine, 2023. URL https://arxiv.org/abs/2301.08745.

Yiqiao Jin, Mohit Chandra, Gaurav Verma, Yibo Hu, Munmun De Choudhury, and Srijan Kumar. Better to ask in english: Cross-lingual evaluation of large language models for healthcare queries. In *Proceedings of the ACM Web Conference 2024*, WWW '24, pp. 2627–2638, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400701719. doi: 10.1145/3589334.3645643. URL https://doi.org/10.1145/3589334.3645643.

Sayash Kapoor and Arvind Narayanan. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9):100804, 2023. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2023.100804. URL https://www.sciencedirect.com/science/article/pii/S2666389923001599.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: a natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Erin LeDell and Sebastien Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020. ICML, 2020.

Mingda Li, Abhijit Mishra, and Utkarsh Mujumdar. Bridging the language gap: Enhancing multilingual prompt-based code generation in llms via zero-shot cross-lingual transfer. *arXiv preprint arXiv:2408.09701*, 2024.

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Alexandre Matton, Tom Sherborne, Dennis Aumiller, Elena Tommasone, Milad Alizadeh, Jingyi He, Raymond Ma, Maxime Voisin, Ellen Gilsenan-McMahon, and Matthias Gallé. On leakage of code generation evaluation datasets, 2024. URL https://arxiv.org/abs/2407.07565.

Micheline Bénédicte Moumoula, Abdoul Kader Kabore, Jacques Klein, and Tegawendé F Bissyande. Evaluating programming language confusion. *arXiv preprint arXiv:2503.13620*, 2025.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng Xin Yong, Hailey Schoelkopf, Xiangru Tang, Dragomir Radev, Alham Fikri Aji, Khalid Almubarak, Samuel Albanie, Zaid Alyafeai, Albert Webson, Edward Raff, and Colin Raffel. Crosslingual generalization through multitask finetuning. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15991–16111, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.891. URL https://aclanthology.org/2023.acl-long.891/.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pp. 485–492, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342063. doi: 10.1145/2908812.2908918. URL https://doi.org/10.1145/2908812.2908918.

Shuyin Ouyang, Dong Huang, Jingwen Guo, Zeyu Sun, Qihao Zhu, and Jie M Zhang. Dscodebench: A realistic benchmark for data science code generation. *arXiv preprint arXiv:2505.15621*, 2025.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Jim Plotts and Megan Risdal. Meta kaggle code, 2023. URL https://www.kaggle.com/ds/3240808.

Nishat Raihan, Antonios Anastasopoulos, and Marcos Zampieri. mHumanEval - a multilingual benchmark to evaluate large language models for code generation. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11432–11461, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.570. URL https://aclanthology.org/2025.naacl-long.570/.

Vikas Raunak, Arul Menezes, Matt Post, and Hany Hassan Awadalla. Do gpts produce less literal translations?, 2023. URL https://arxiv.org/abs/2305.16806.

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

L. Sasse, E. Nicolaisen-Sobesky, J. Dukart, S. B. Eickhoff, M. Götz, S. Hamdan, V. Komeyer, A. Kulkarni, J. M. Lahnakoski, B. C. Love, F. Raimondo, and Kaustubh R. Patil. Overview of leakage scenarios in supervised machine learning. *Journal of Big Data*, 12(1), May 2025. ISSN 2196-1115. doi: 10.1186/s40537-025-01193-8. URL http://dx.doi.org/10.1186/s40537-025-01193-8.

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

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, Shengzhu Yin, Minhui Zhu, Kilian Lieret, Yanxin Lu, Genglin Liu, Yufeng Du, Tianhua Tao, Ofir Press, Jamie Callan, Eliu Huerta, and Hao Peng. Scicode: A research coding benchmark curated by scientists. *arXiv preprint arXiv:2407.13168*, 2024.

Anton Vakhrushev, Alexander Ryzhkov, Maxim Savchenko, Dmitry Simakov, Rinchin Damdinov, and Alexander Tuzhilin. Lightautoml: Automl solution for a large financial services ecosystem, 2022. URL https://arxiv.org/abs/2109.01528.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025. URL https://arxiv.org/abs/2407.16741.

Zhiruo Wang, Grace Cuenca, Shuyan Zhou, Frank F Xu, and Graham Neubig. Mconala: A benchmark for code generation from multiple natural languages. *arXiv preprint arXiv:2203.08388*, 2022.

Chenyang Yang, Rachel A Brower-Sinning, Grace A. Lewis, and Christian Kästner. Data leakage in notebooks: Static detection and better processes, 2022. URL https://arxiv.org/abs/2209.03345.

Xin Zhou, Martin Weyssow, Ratnadira Widyasari, Ting Zhang, Junda He, Yunbo Lyu, Jianming Chang, Beiqi Zhang, Dan Huang, and David Lo. Lessleak-bench: A first investigation of data leakage in llms across 83 software engineering benchmarks, 2025. URL https://arxiv.org/abs/2502.06215.

Marc-André Zöller and Marco F. Huber. Benchmark and survey of automated machine learning frameworks. *J. Artif. Int. Res.*, 70:409–472, May 2021. ISSN 1076-9757. doi: 10.1613/jair.1.11854. URL https://doi.org/10.1613/jair.1.11854.

A AutoML Frameworks Discussion

There is a large scope of AutoML frameworks that apply different techniques and achieve variable results. For instance, one of the most popular methods involves ML-pipeline and parameter optimization via either Grid Search and Random Search (H2O AutoML (LeDell & Poirier, 2020)) or Bayesian (Auto-sklearn Feurer et al. (2015)) or genetic algorithms (TPOT (Olson et al., 2016)) methods.

One of the most advanced methods in AutoML is Neural Architecture Search (NAS) (Elsken et al., 2019) that automatically designs neural network topologies. Frameworks such as DARTS (Liu et al., 2019) and ENAS (Pham et al., 2018) have shown significant promise in discovering novel, optimized architectures that often outperform manually designed models for specific tasks. It includes three core components: the search space for potential architectures, the optimization methods for discovering the best-performing architecture, and the model evaluation techniques. By automating the neural architecture design process, NAS can generate more efficient and specialized models, contributing to significant advancements in AutoML.

However, while NAS has achieved remarkable performance, it currently provides limited insights into why certain architectures perform well or how similar architectures are across independent runs. Furthemore, it requires enormous computational resources and accurate design of the search space Liu et al. (2019) that makes it challenging for the ML-research.

B LLM Pipeline For Description Creation

As stated in 3.1 200 competitions have been processed with the use of LLM pipeline which is depicted in detail in Figure 6. Subsequently, 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% (Figure 7, Algorithm 1).

Algorithm 1 Scoring-Refinement Algorithm

Initial description x_0, input code c, model $\mathcal{M}$, prompts $\{p_{score}, p_{refine}\}$ $x_t \leftarrow x_0$ Initialize with the given description iteration $t = 0, 1, \ldots$ $score_t \leftarrow \mathcal{M}(p_{score} \,\|\, x_t)$ Evaluate description with scoring prompt $score_t \in \{C, D\}$ **break** Stop if score is satisfactory $score_t \in \{A, B\}$ $x_{t+1} \leftarrow \mathcal{M}(p_{refine} \,\|\, c \,\|\, x_t \,\|\, score_t)$ Refine using code, description, and score

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 2.

If annotators disagree by one point, we conservatively 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.

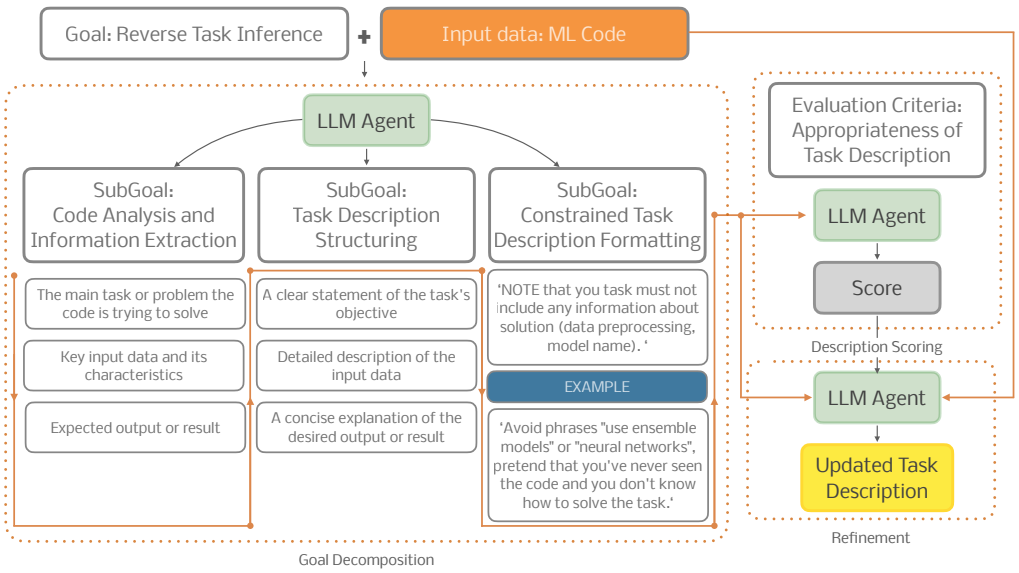

Figure 6: Code-Based problem statement generation framework. The scheme incorporates three LLM agents. The first agent inputs the ML code to infer the task description from it through sequential subgoals. The second agent evaluates the quality of the inferred description based on predefined scoring criteria. The third agent receives the ML code along with the score and updates the description if necessary.

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

Table 2: 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.

Prompt example

Translate text into **{target_language}**. Infinitive forms that stand apart, if any, should be translated as the imperative mood: **{text}**

Figure 8: Example of the prompt used in translation experiments.

C Domain Extraction Prompt

Figure 9 demonstrates the prompt that has been given to GPT-3.5-turbo model to derive domain tag for each competition. The number of competitions in each domain is presented in Figure 10.

D Competition Selection Criteria

Table 3 gives details on selection criteria according to which competitions have been chosen.

Table 3: Task selection criteria for GenML²Bench

Criterion	Description
Dataset size	$\leq$15 GB to ensure feasibility under memory/runtime limits.
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 license	The dataset's license doesn't restrict its inclusion in our benchmark.

E Translation Prompt

Since some fields include imperatives (e.g., *Develop a model*, *Create an agent*), it has to be defined explicitly in a prompt (Figure 8) to use imperative mood, otherwise the model have translated english imperatives, which have the same form as verbs not in imperative mood, mainly as infinitives:

F Annotator Details

As mentioned in section 3.4, validation of the translation has been conducted with the help of native speakers. They possess backgrounds in Computer Science and/or IT. Each translator has been assigned three forms with 30 questions each, getting no monetary compensation.

You are given competition name, data card and description of Kaggle competition. You need to identify the domain that the task belongs to in the given competition.

Competition name: Crime_Learn

Description: Develop a predictive model to estimate the rate of violent crimes per population in a given area based on specific features. The input consists of two datasets, one for training and one for testing, with the target variable being 'ViolentCrimesPerPop'.

Data card: In this competition you will use the sample US crime data for predicting 'ViolentCrimesPerPop'. train.csv – the training dataset.

Figure 9: Example of question block in Google Form for Romanian language

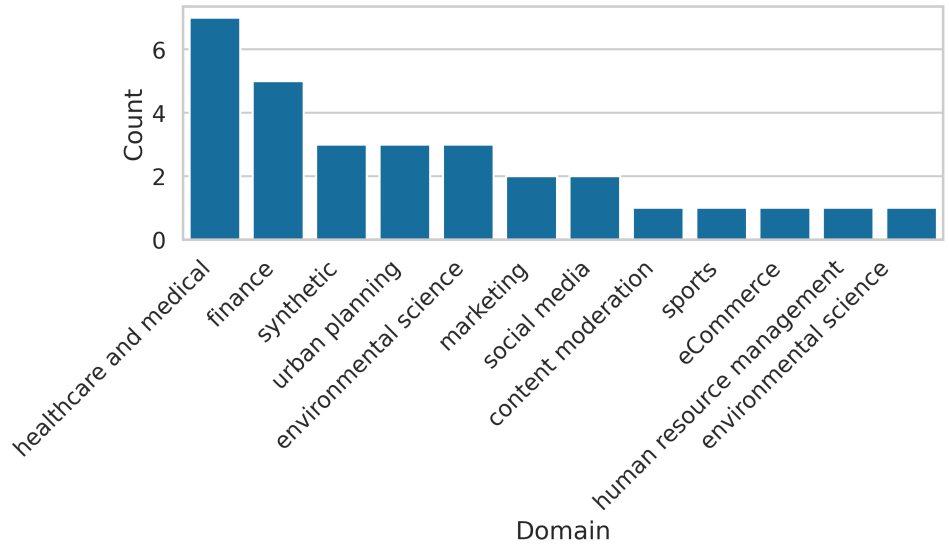

Figure 10: Distribution of competitions over domains

G Validation Results

Figure 11 indicates that nearly two-thirds of responses within each language are both judged natural and semantically equivalent, with Romanian and Kazakh exhibiting the lowest proportions among the evaluated language. This pattern is consistent with evidence that GPT-based translation quality degrades for low-resource languages, which typically have fewer native speakers and, thus, less training data available (Hendy et al., 2023).

Figure 12 (A) further shows that GPT systems predominantly produce natural translations without semantic distortion, with only 1.3 % of all outputs rated neither natural nor semantically similar to the source and with almost 4 % translations sounding natural but conveying other meaning. The share of labeled "NO and YES" suggests that, while models preserve meaning, they often employ more varied and less concise phrasings in the target language.

Figure 12 (B-D) shows that the proportion of responses indicating that translations both preserve meaning and sound natural is highest in the domain texts (85.6%). This outcome may be attributed to the brevity of the source material, as the texts contained no more than three words. Shorter inputs are generally easier to render accurately, which may explain why the models achieved stronger performance in this setting.

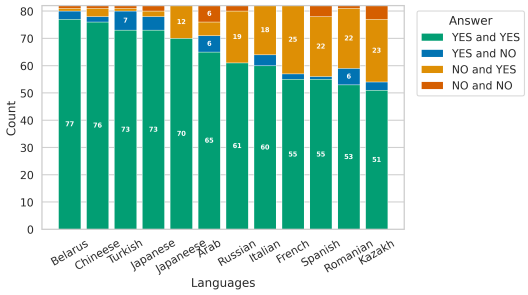

Figure 11: Distribution of response types within each language

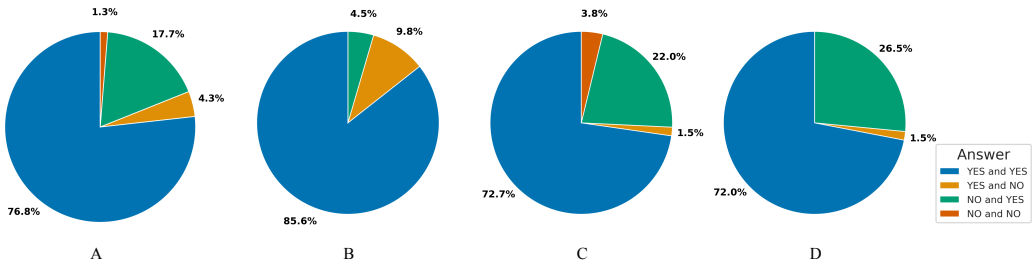

Figure 12: Distribution of translation evaluation outcomes. (A) Overall distribution across all languages; (B–D) distributions for the evaluation of translations of domains (B), data cards (C), and task descriptions (D), respectively.

H FORM EXAMPLE

In Fig.13 there is an example of one question block in a form, which requires a native of Romaninan validate the translation of competition description.

Translated version:

Dezvoltați un model predictiv pentru a anticipa probabilitatea mortalității în spital pentru pacienți. Seturile de date includ diverse caracteristici legate de pacienți la momentul internării în spital. Obiectivul este de a prezice cu acuratețe probabilitatea mortalității în spital pentru fiecare pacient din setul de testare.

Original version:

Develop a predictive model to forecast the likelihood of hospital mortality for patients. The datasets include various features related to the patients upon hospital admission. The objective is to predict the probability of hospital mortality for each patient in the test set accurately.

Does the translated text (1) sound native and (2) convey the same meaning as the original text?

- YES and YES
- NO and YES
- YES and NO
- NO and NO

If there is at least one NO in the answer, please suggest your own version:

Figure 13: Example of question block in Google Form for Romanian language

I RESULTS

This appendix provides detailed benchmark results across different data types, domains, and evaluation metrics. The results highlight the comparative performance of leading large language models (LLMs) in diverse tasks. Tables 5, 6, and 7 summarize the top three models for each category.

J DATA LEAKAGE

Figure 14 shows data leakage assessment process.

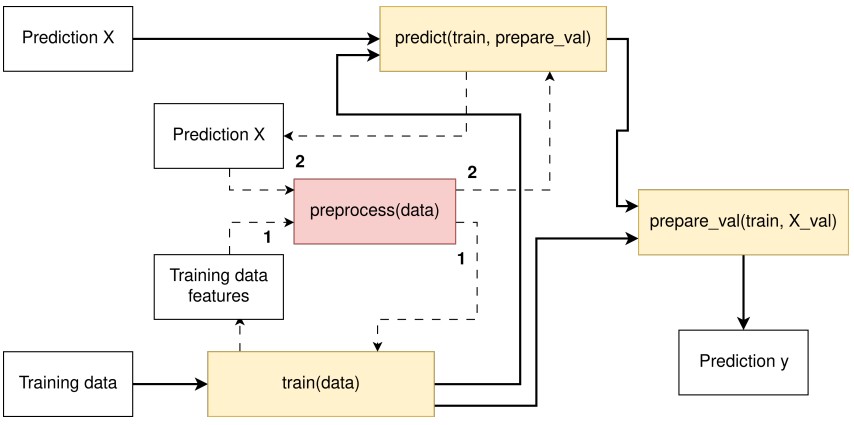

Figure 14: Code flow diagram of a false-positive data leakage

Table 4: Sample results of generated ML code validated on the Kaggle platform detailed by domains. For each model-domain-language triple, the median percentiles based on leaderboard rankings are presented. Lower percentile values indicate better solution quality. The best results are highlighted in bold. The symbol '-' indicates that the model failed to successfully complete the task, resulting in the generation of non-functional code.

Framework	Model	Domain	Category	Arab	Belarus	Chinese	English	Italian	Japanese	Kazakh	Polish	Romanian	Spanish	Russian	French	Turkish
AIDE[1]	41.8cmgpt-oss-120	image	Categorization	**1**	23	**1**	**1**	**1**	**1**	**1**	18	**1**	**1**	48	**1**	48
		tabular	Classification	69	69	70	68	**47**	60	**32**	69	70	70	79	**23**	100
		tabular	Regression	54	50	58	36	47	53	58	59	**22**	**30**	43	70	**32**
		text	Classification	98	99	99	95	98	97	95	**84**	97	99	96	98	**1**
	41.8cmgemini 2.5 flash	image	Categorization	55	100	32	67	**1**	47	**1**	49	**1**	55	48	5	**1**
		tabular	Classification	84	**48**	100	90	95	100	**30**	100	**32**	80	89	55	68
		tabular	Regression	72	69	30	**16**	**8**	19	72	23	86	**10**	33	34	64
		text	Classification	99	47	**2**	**1**	49	52	95	50	71	**1**	65	95	44
	41.8cmgpt-4.1-mini	image	Categorization	**1**	**1**	96	56	67	98	**1**	37	95	49	**1**	**1**	56
		tabular	Classification	**68**	90	70	79	**64**	92	89	84	76	84	99	80	**69**
		tabular	Regression	47	39	76	40	47	54	**21**	**32**	**21**	76	33	40	56
		text	Classification	88	97	95	81	72	99	53	**50**	95	**1**	59	98	**49**
ML-Master[2]	31.8cmgpt-4.1-mini + deepseek-r1	tabular	Classification	26	26	58	26	59	**3**	22	26	26	**3**	26	61	**19**
		tabular	Regression	**3**	**3**	**3**	**3**	**3**	77	**3**	**3**	**3**	44	**3**	**3**	**3**
		text	Classification	100	-	44	**18**	52	-	49	46	-	67	-	**18**	-
	31.8cmgpt-oss:120b + qwen3-coder:30b	tabular	Classification	28	28	**14**	24	**22**	62	**13**	63	62	28	64	62	28
		tabular	Regression	18	28	44	19	**3**	34	29	16	18	**9**	46	**17**	34
		text	Classification	92	**22**	31	42	100	39	**23**	34	100	**22**	34	32	52

Table 5: Top models by data type.

Data Type	Rank 1	Score	Rank 2 / Rank 3
Image	gemini-2.5-flash (aide)	0.7313	gpt-oss-120b (0.7151), gpt-4.1-mini (0.5572)
Tabular	gpt-oss-120b+qwen3-30b (ml-master)	2673.24	gpt-oss-120b (2219.75), gemini-2.5-flash (1698.80)
Text	gpt-oss-120b (aide)	0.9857	gpt-4.1-mini+deepseek-r1 (0.9540), gemini-2.5-flash (0.93

Table 6: Top models by domain.

Domain	Rank 1	Score	Rank 2 / Rank 3
Content moderation	gpt-oss-120b	0.9857	gpt-4.1-mini (0.9763), gemini-2.5-flash (0.9682)
Crime prediction	gemini-2.5-flash	380.99	gpt-4.1-mini (376.02), gpt-oss-120b (373.32)
Data science	gpt-4.1-mini	0.9733	gemini-2.5-flash (0.9659), gpt-oss-120b (0.9482)
eCommerce	gpt-4.1-mini+deepseek-r1	0.9304	gpt-4.1-mini (0.9288), gemini-2.5-flash (0.9272)
Education	gpt-oss-120b+qwen3-30b	12.91	gpt-4.1-mini (6.44), gpt-oss-120b (6.30)
Environmental sci.	gemini-2.5-flash	0.9281	gpt-oss-120b (0.8779), gpt-4.1-mini (0.6603)
Finance	gpt-oss-120b	7.32	gpt-4.1-mini (6.48), gemini-2.5-flash (4.56)
Healthcare/Medical	gemini-2.5-flash	1.85	gpt-4.1-mini (1.38), gpt-oss-120b (1.22)
Urban planning	gpt-oss-120b+qwen3-30b	3994.47	gpt-4.1-mini+deepseek-r1 (3513.81), gpt-oss-120b (3131.19)

Table 7: Top models by evaluation metric (normalized).

Metric	Rank 1	Score	Rank 2 / Rank 3
AUC	gpt-4.1-mini+deepseek-r1	0.9076	gpt-oss-120b (0.9047), gemini-2.5-flash (0.8963)
Accuracy	gemini-2.5-flash	0.8766	gpt-oss-120b (0.8317), gpt-4.1-mini (0.7736)
F-score	gpt-oss-120b	0.9595	gpt-4.1-mini+deepseek-r1 (0.9540), gpt-oss-120b+qwen3-30b (0.9502)
R^2	gpt-oss-120b	0.9285	gemini-2.5-flash (0.9281), gpt-4.1-mini (0.9279)
MAP	gpt-4.1-mini	0.2255	gpt-oss-120b (0.2197), gemini-2.5-flash (0.0000)
MAE	gpt-4.1-mini+deepseek-r1	1.0000	gpt-oss-120b+qwen3-30b (0.5618), gpt-oss-120b (0.5508)
MSE	gemini-2.5-flash	0.5728	gpt-4.1-mini (0.5538), gpt-4.1-mini+deepseek-r1 (0.5252)