# ML$^2$B: Benchmarking LLMs on Cross-Lingual ML Pipeline Generation

Ekaterina Trofimova[1,*], Zosya Shamina[2], Maria Selifanova[2], Artem Zaitsev[2], Remi Savchuk[2], Maxim Minets[2], Daria Ozerova[2], Emil Sataev[2], Denis Zuenko[2] and Andrey Ustyuzhanin[1,3]

[1]*Constructor Labs, Campus Ring 1, 28759 Vegesack, Bremen, Germany*
[2]*Independent researcher*
[3]*National University of Singapore, 21 Lower Kent Ridge Rd, University Hall, 119077, Singapore*

**Abstract**

We introduce ML$^2$B, the first benchmark for evaluating cross-lingual task comprehension in end-to-end ML pipeline generation by large language models. Despite growing global AI adoption, no systematic evaluation exists for ML pipeline generation beyond English task descriptions. ML$^2$B addresses this gap with 35 Kaggle competitions spanning tabular, text, and image domains, translated into 14 languages by native-speaker researchers with ML expertise, yielding 490 task-language pairs. To ensure evaluation integrity, the benchmark incorporates 10 private competitions without publicly available solutions and employs network-isolated evaluation infrastructure restricting runtime access to essential ML resources. We provide standardized evaluation protocols, an AutoGluon algorithmic baseline, and comprehensive failure mode analysis. Experiments with frontier models (GPT-4.1-mini, GPT-OSS-120b, Gemini-2.5-Flash) reveal that cross-lingual performance degradation is highly task-dependent rather than following traditional resource-availability hierarchies, with gaps ranging from language advantages to severe degradation depending on competition characteristics. These findings challenge conventional assumptions about multilingual model capabilities and underscore the necessity of systematic cross-lingual evaluation for ML pipeline generation. We open-source the benchmark, baselines, and evaluation infrastructure at https://github.com/enaix/ml2b.

**Keywords**

Multilingual machine learning, large language models, cross-lingual representation learning, code generation, machine learning work-flows, benchmark dataset



## 1. Introduction

Machine learning (ML) has become a fundamental component across various domains, motivating the development of AutoML frameworks to automate pipeline selection [1] and LLM-based code generation benchmarks such as MLE-Bench [2], DA-Code [3], and Weco-Kaggle [4].

All existing benchmarks are limited to English. LLMs exhibit systematic performance degradation in other languages [5, 6], with substantial drops for low-resource languages [7, 8]. Meanwhile, ML research is global: models must interpret problem descriptions in diverse languages while producing executable code. Current benchmarks cannot measure this cross-lingual robustness.

Another challenge arises from *benchmark data leakage*, when benchmark data is also present in the LLM training data [9]. This issue is particularly important, as the model may overperform in particular benchmark tasks. In the worst-case scenario, this may lead to the affected benchmark competitions being inconclusive [10]. A similar form of data leakage happens when unintended information about

*GeCoIn 2026: Generative Code Intelligence Workshop, co-located with the 35th International Joint Conference on Artificial Intelligence (IJCAI-ECAI 2026), August 15–17, 2026 — Bremen, Germany*
*Corresponding author.
ekaterina.trofimova@constructor.org (E. Trofimova); zosyasham@gmail.com (Z. Shamina); mariaselifanova1@gmail.com (M. Selifanova); arvlzaitsev@gmail.com (A. Zaitsev); enaix@protonmail.com (R. Savchuk); m.maxim.v.01@gmail.com (M. Minets); dozerova01@gmail.com (D. Ozerova); emils99mail@gmail.com (E. Sataev); neron.v2@gmail.com (D. Zuenko); austyuzhan@constructor.university (A. Ustyuzhanin)
0000-0001-5436-8511 (E. Trofimova); 0009-0000-2870-4403 (Z. Shamina); 0009-0009-7065-0512 (M. Selifanova); 0009-0002-0286-6484 (A. Zaitsev); 0009-0007-9083-0292 (R. Savchuk); 0009-0004-5895-1391 (M. Minets); 0009-0002-5423-6409 (D. Ozerova); 0009-0000-2891-0792 (E. Sataev); 0000-0002-6148-5107 (D. Zuenko)

**Table 1**
Comparison of ML$^2$B with related benchmarks: leakage detection, tasks without public solutions (reducing training-data contamination), assessment of specific ML capabilities, and grading in a network-isolated container.

| Benchmark | Multilingual | Leakage Prevention | Private Competitions | ML Capabilities | Isolated Grading |
|---|---|---|---|---|---|
| ML$^2$B (ours) | ✓ | ✓ | ✓ | ✗ | ✓ |
| MLE-Bench | ✗ | ✗ | ✗ | ✗ | ✗ |
| ML-Dev-Bench | ✗ | ✗ | ✓ | ✓ | ✗ |
| MLAgentBench | ✗ | ✗ | ✗ | ✓ | ✗ |
| DSCodeBench | ✗ | ✗ | ✗ | ✓ | ✗ |

the test set appears in training data. Leakage can artificially inflate performance and produce unreliable results [11, 12, 13]. It is also pervasive in real-world ML code [14].

To address these shortcomings, we introduce ML$^2$B, the first benchmark for evaluating LLMs on generating complete ML pipelines from multilingual natural language descriptions. Our key contributions are:

1. **Multilingual benchmark.** 35 Kaggle competitions translated into 14 natural languages, yielding 490 evaluation instances.
2. **Private competitions.** 10 competitions without publicly available solutions, reducing the likelihood of training data contamination.
3. **Robust code evaluation.** Network-isolated grading, modular submission format, and static leakage detection.

Beyond the language dimension, ML$^2$B differs from MLE-Bench and DA-Code methodologically: agent code is executed directly via AST-based recompilation rather than scored as a submission file, the modular format structurally separates training from prediction data, and grading runs in a network-isolated container (Table 1).

## 2. Related work

### 2.1. ML Code Generation and AutoML

Recent benchmarks evaluate LLM capabilities for ML code generation across different scopes. DSCodeBench [15] and DS-1000 [16] assess snippet-level code from GitHub and StackOverflow. Full-pipeline benchmarks include DA-Code [3], Weco-Kaggle [4], and MLE-bench [2], which leverage Kaggle competitions. MLE-bench evaluates agents on 75 tasks; ML-Dev-Bench [17] provides 30 tasks for debugging and API integration; MLAgentBench [18] offers 13 research-oriented scenarios. All are English-only.

Contemporary LLM-based agents for ML engineering employ iterative refinement strategies. AIDE [4] uses solution space tree search, outperforming traditional AutoML frameworks like LightAutoML [19] and OpenHands [20] on Kaggle tasks. Recent multi-agent frameworks like ML-Master [21] and R&D-Agent [22] achieve superior performance but require substantially larger models (GPT-4.1, GPT-5).

### 2.2. Multilingual Code Generation

Multilingual datasets for code generation remain scarce. MCoNaLa [23] consists of intents for code generation, which are further rewritten by human annotators, and code snippets in Python. RoCode [24] offers Romanian programming problems with Python/C++ solutions. MBPP-Translated [25] extends MBPP to five languages using machine translation. mHumanEval [8] supports 204 languages, with expert translation for 15, across 25 programming languages. However, none target end-to-end ML pipelines.

Several studies show LLM performance depends strongly on prompt language. Bang et al. [26], Ahuja et al. [27], Muennighoff et al. [28], and Raihan et al. [8] report substantial drops for low-resource

languages. Moumoula et al. [29] analyze 13 programming and 23 natural languages, showing that non-Latin scripts further degrade performance. While targeted multilingual fine-tuning can partially mitigate these issues [30, 31], current ML code-generation benchmarks provide no means to measure such cross-lingual robustness.

# 3. The ML$^2$B benchmark

ML$^2$B provides structured, succinct metadata and task descriptions designed for efficient LLM processing, unlike MLE-Bench [2], which uses verbose descriptions sourced directly from Kaggle. Structured prompts achieve better performance on 5 of 10 shared competitions and perfect stability (3/3 pass rate) versus obfuscated format failures on 5 competitions, confirming that clear task descriptions aid comprehension without enabling memorization-based shortcuts (Appendix A). The benchmark comprises 35 competitions spanning 14 languages (490 evaluation instances), with 10 private competitions to mitigate data contamination (Appendix B).

## 3.1. Benchmark task selection

The ML$^2$B benchmark builds on the Code4ML 2.0 dataset [32], which provides standardized task descriptions for Kaggle competitions. We first identify all competitions meeting our inclusion criteria and released after 2020. From these, we manually validate and select 22 tasks with publicly available Kaggle code. Selected competitions satisfy the following conditions: (1) the competition is closed, ensuring fixed data and leaderboards; (2) the dataset is publicly downloadable; (3) the evaluation metric is documented; (4) the evaluation is reproducible with complete metadata; and (5) submissions follow a tabular prediction format.

As Code4ML 2.0 covers competitions up to 2024, we additionally include three public competitions released in 2025, resulting in 25 public tasks. To assess generalization to unseen problems, we further add 10 private competitions without publicly available code, i.e., finished, closed Kaggle competitions with no public solution code, no forum discussions, and limited web indexability (hosted by university courses or small organizations). These retain public leaderboards, enabling consistent baseline evaluation (Section 3.5), but lack solution code and forums, reducing the likelihood of inclusion in LLM training data. Task descriptions are manually summarized from the corresponding Kaggle pages.

All datasets and task descriptions are sourced from Kaggle competition pages and released under Kaggle's standard competition license or permissive Creative Commons licenses (CC BY 4.0 or CC BY-NC 4.0), as specified on each competition page. These licenses allow redistribution with attribution, with CC BY-NC 4.0 restricting commercial use.

Unlike Code4ML 2.0, ML$^2$B provides a manually curated data card for each task in addition to the data source link. Data cards and task descriptions are manually reviewed to ensure clarity and to prevent information leakage that could confer an unfair advantage. In particular, details related to Kaggle submission formats and test files are removed, as test data lack labels and are irrelevant to our executable-code generation setting. Each task also includes its evaluation metric and metric type, following the taxonomy of [33].

## 3.2. Benchmark Characteristics

Table 2 summarizes ML$^2$B coverage. Competitions span 9 domains (finance, healthcare and medical, manufacturing and quality, environmental studies, media studies, social studies, urban studies, marketing, synthetic data), ensuring broad coverage (Appendix C). Evaluation metrics include 5 higher-is-better (AUC, F1, MAP, $R^2$, accuracy) and 3 lower-is-better (RMSE, MAE, log-loss) metrics. Overall, both the full and chosen sets share relatively similar characteristic distributions, which supports using the subset to present our results sustainably.

**Table 2**
Competition characteristics on full and sub-set also grouped by public vs. private partition. The private split is identical for the full set and the chosen one.

| Partition | Set | Size | **Data type** (tab / text / img) | **Metric direction** (higher / lower) |
|---|---|---|---|---|
| Public | Full set | 25 | 20 / 2 / 3 | 14 / 11 |
| | Chosen set | 5 | 3 / 1 / 1 | 3 / 2 |
| Private | Full/chosen set | 10 | 8 / 1 / 1 | 5 / 5 |

## 3.3. Metadata and Structure

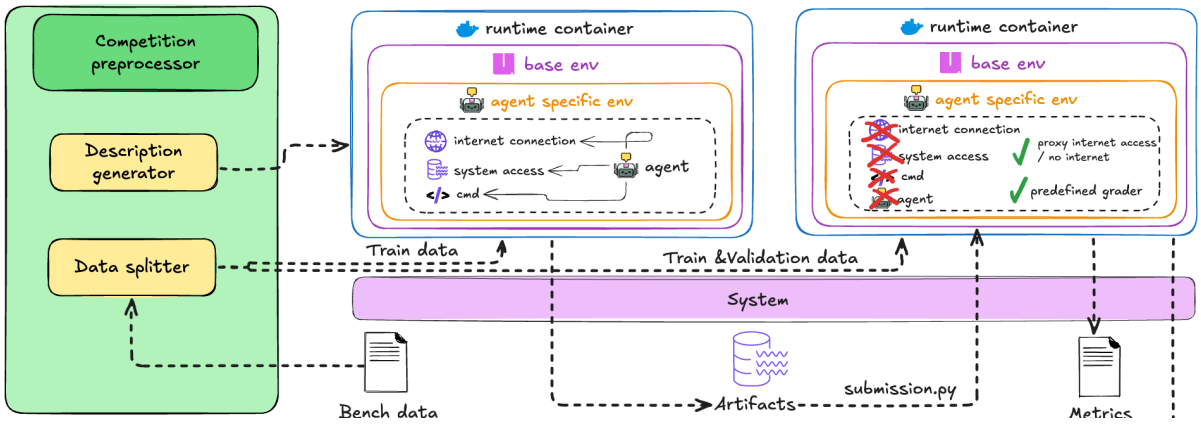



**Figure 1:** Structure of the ML$^2$B benchmark. Competition preprocessor prepares task descriptions and data, runtime container generates the solution while evaluation container executes the solution code

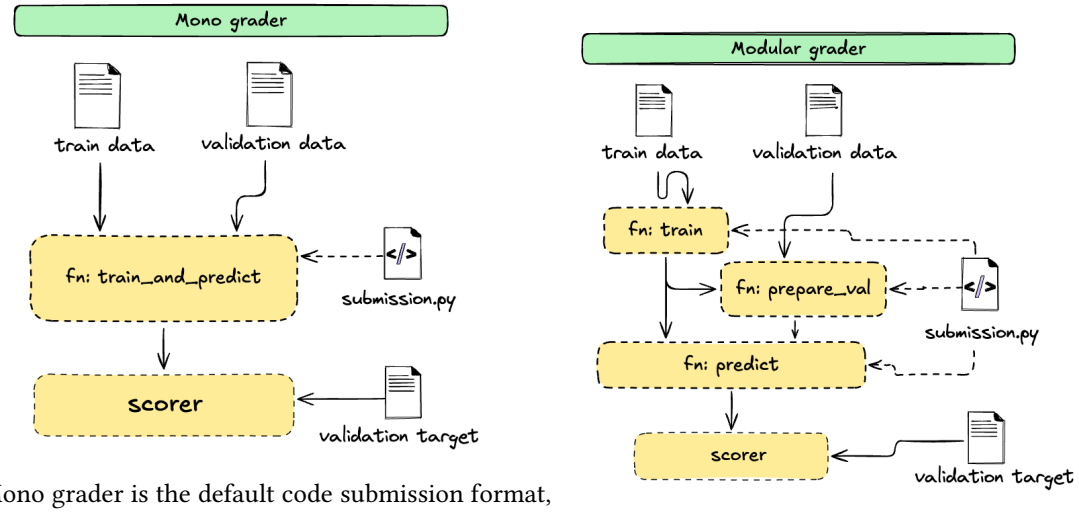
The benchmark consists of 3 main components (Figure 1), which include the main competition preprocessor, Docker agent runtime, and the submission code grader. The competition preprocessor is responsible for task description generation (Appendix D) and competition data preparation, the agent runtime manages the AutoML agent, and the code grader evaluates a metric for the submission code in an isolated environment.

Both the agent and code grader are executed inside the Docker environment. The agent Docker image is built from the common runtime image, and both the agent and grader containers are built from the same 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 the event of a misconfigured or malicious script being submitted.

Agent runtime permits full internet access during code generation. The evaluation grader operates behind a forward proxy restricting access to essential ML infrastructure (HuggingFace, PyTorch, S3), blocking Kaggle and GitHub to prevent solution retrieval at runtime.

Instead of the Kaggle-style submission format, which consists of a single submission file, the code grader 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 an 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 an Abstract Syntax Tree (AST) transformation. Then, the recompiled submission code is executed 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.

The benchmark supports two submission formats: single-function submission format `MONO_PREDICT` (Figure 2a) and modular submission format `MODULAR_PREDICT` (Figure 2b). `MODULAR_PREDICT` con-




(a) Mono grader is the default code submission format, where a single-function solution returns the prediction

(b) Modular grader is the introduced format which isolates the train data from the validation data

**Figure 2:** Code flow diagram of agent solution submission formats

sists of three functions `train`, `prepare_val`, and `predict`, which sequentially train the model, prepare the prediction data, and predict the result. The purpose of such a prediction format is to reduce the chance of preprocessing leakage. Preprocessing leakage is the type of data leakage when both training and test data are processed together Yang et al. [12], Apicella et al. [11]. 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. In order to assess the presence of data leakage in submission code, static leakage analysis was performed using the `leakage-analysis` tool [12]. This tool finds potential relations between the variables and outputs lines of code, causing potential data leakage. Leakage analysis is performed separately from the benchmark pipeline. Static leakage analysis finds potential leakage in $<10\%$ of submissions in `MODULAR_PREDICT` format, with 4% remaining after excluding false positives in `train`-only and single-argument functions (Appendix E and Figure 3).

### 3.4. Multilingual Expansion

To obtain a multilingual corpus, we translate the *domain*, *description*, and *data card* fields into target languages (Appendix F). Other fields do not require translation since they convey universally recognized entities. Following Jiao et al. [34], we use GPT-4o over commercial translators such as Google Translate and DeepL, as it matches or exceeds their quality for most language pairs.

After automatic translation, texts undergo manual validation by bilingual annotators with ML experience and at least a bachelor's degree in a computer-science–related field. Annotators are researchers from the authors' current and former affiliated institutions across multiple countries, ensuring authentic native-speaker validation; all participated voluntarily as academics collaborators and received co-authorship or acknowledgment commensurate with their contributions. We employ single-annotator validation per language, as GPT-4o's demonstrated translation quality means that annotators primarily perform error detection rather than de novo translation.

To structure validation, each annotator has received three Google Forms (one for each translated field). Each form presents the English source, the translation, and a question assessing whether the text sounds natural and preserves the original meaning. If the answer is "No," the annotator has supplied a corrected version (Appendix G).

The final benchmark includes translations in Arabic, Belarusian, Chinese, French, Italian, Japanese,

Kazakh, Polish, Romanian, Russian, Spanish, Turkish and Ukrainian, spanning diverse language families (Romance, Slavic, Semitic, Turkic, Sino-Tibetan, Japonic), writing systems (Latin, Cyrillic, Arabic, CJK), and resource levels. Language selection is constrained by the availability of qualified annotators. Translation quality is verified via back-translation BLEU (mean >0.35 for all language-field combinations) and inter-annotator agreement on a subset of 15 competitions (Gwet's AC1 [35] >0.80 for meaning preservation across all languages; see Appendix H).

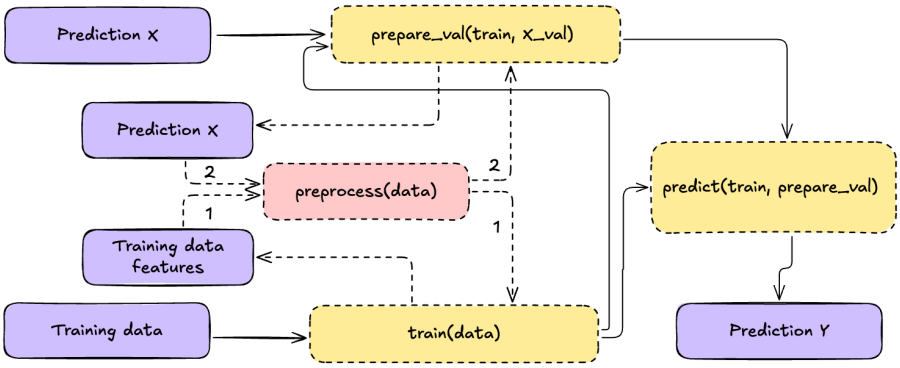



**Figure 3:** Code flow diagram of a false-positive data leakage, where the preprocessing function is called sequentially

### 3.5. Evaluation

Evaluating ML code generation across heterogeneous competitions requires careful metric design. Competitions use diverse evaluation metrics (e.g., RMSE, log-loss, F1-score, AUC) with incomparable scales and directionality. We address this through normalized performance measurement.

For each competition $p$, we establish two complementary in-benchmark baselines. The primary baseline uses LLM-assisted manual development: we iteratively prompt Gemini 3 Pro for solution approaches, implement standard ML pipelines (e.g., LightGBM with basic feature engineering), and validate on the 20% validation split. The baseline score $b_p$ represents competent, manually-validated ML practice that handles heterogeneous task types uniformly. As a fully reproducible algorithmic reference point, we additionally provide an AutoGluon [36, 37] baseline. We prefer AutoGluon over Auto-sklearn because it natively supports all data types in ML$^2$B (tabular, text, and image).

For a model with score $s_p$ on competition $p$, we compute the performance ratio:

$$r_p = \frac{e_p}{e_{b,p}} \tag{1}$$

where $e_p$ and $e_{b,p}$ are the error rates of the model and baseline, respectively, defined as $e = 1 - s$ for higher-better metrics (e.g., F1, AUC) and $e = s$ for lower-better metrics (e.g., RMSE, log-loss). Thus, $r_p < 1$ indicates better-than-baseline performance, $r_p = 1$ parity, and $r_p > 1$ worse performance.

Following Dolan and Moré [38], we define the performance profile as the cumulative distribution of performance ratios:

$$\rho(\tau) = \frac{1}{N} \sum_{p=1}^{N} \mathbb{I}[r_p \leq \tau] \tag{2}$$

representing the fraction of tasks where the model achieves performance within factor $\tau$ of baseline.

The standard Dolan-Moré formulation integrates $\rho(\tau)$ over $\tau \geq 1$ to measure worse-than-best performance. In our setting, where $r_p < 1$ indicates *better*-than-baseline performance, we adapt the integration domain to $\tau \in [0, 1]$, yielding the standard area under the profile:

$$\text{AUP}_{\text{std}} = \int_0^1 \rho(\tau)\, d\tau. \tag{3}$$

This formulation emphasizes the region $r_p \leq 1$ (baseline-meeting or better performance). To emphasize different performance regimes, we use weighted AUP:

$$\text{AUP}_w = \frac{\int_0^1 \rho(\tau) \cdot w(\tau)\, d\tau}{\int_0^1 w(\tau)\, d\tau}. \tag{4}$$

We report $\text{AUP}_{-1/\ln \tau}$ with $w(\tau) = -\frac{1}{\ln \tau}$, which emphasizes near-baseline distinctions ($r_p \approx 1$) and best distinguishes models that consistently meet baseline from those that occasionally succeed (formal definitions and numerical implementation details in Appendix I).

## 4. Experiments and Results

We evaluate three LLMs: GPT-4.1-mini, GPT-OSS-120b, and Gemini-2.5-Flash, using two agent frameworks: AIDE [4] and ReAct [39]. AIDE experiments use the standard configuration during code generation. For GPT-OSS-120b, we additionally evaluate a modified single-agent ReAct framework equipped with standard file system interaction capabilities (read, write, create) and the ability to execute Python and Bash commands. While the base ReAct agent cannot solve complex benchmark tasks requiring strictly structured Python files with predefined validation signatures, we address this limitation through two enhancements: tool execution timeouts prevent system freezing during prolonged operations, and the agent constructs structured command execution plans ensuring complete pipeline execution before termination.

Each model-framework configuration is executed 3 times per task with a 3-hour wall-clock time limit per run, yielding 630 runs per configuration (15 competitions × 14 languages × 3 runs). If executed sequentially in a single process, this amounts to 1,890 computing hours per configuration. A complete benchmark evaluation across all 4 model-framework configurations totals 2,520 runs and 7,560 sequential computing hours, though this duration can be reduced proportionally through parallel execution. All code execution is conducted on consumer-grade hardware representative of typical research environments: a workstation equipped with 64GB RAM and a single NVIDIA GeForce RTX 2080 Ti GPU (12GB VRAM). The average cost per configuration run ranges from 6.36 USD for GPT-OSS-120b to 38.48 USD for Gemini-2.5-Flash.

AutoGluon produces valid baselines on 13 of 15 competitions; two failures (Multi-label Classification, DataLab Cup) exceed GPU memory. Both baselines use the same 80/20 train/validation split (random seed 42) as all model evaluations and are available on GitHub. Table 3 confirms that AUP rankings and core findings are consistent across both normalizations.

**Table 3**
$\text{AUP}_{-1/\ln \tau}$ ± std / Pass Rate (%) under both baseline normalizations on the ML$^2$B chosen set (15 competitions, 14 languages, 3 runs). Pass rates are baseline-independent.

| Model / Framework | LLM-assisted baseline | AutoGluon baseline |
|---|---|---|
| GPT-4.1-mini (AIDE) | .768 ± .112 / 61 | .736 ± .106 / 61 |
| Gemini-2.5-Flash (AIDE) | .732 ± .091 / 76 | .775 ± .075 / 76 |
| GPT-OSS-120b (AIDE) | .630 ± .083 / 81 | .616 ± .068 / 81 |
| GPT-OSS-120b (ReAct) | .705 ± .081 / 76 | .732 ± .094 / 76 |

Table 4 presents the main results across all configurations. GPT-4.1-mini achieves the highest average AUP (0.768 ± 0.112) but with the lowest average pass rate (61%), showing strong ranking ability but difficulty with exact solutions. Gemini-2.5-Flash demonstrates more balanced performance with high AUP (0.732 ± 0.091) and the best average pass rate (76%). Among open-source models, GPT-OSS-120b in the AIDE framework shows a competitive AUP (0.630 ± 0.083) with a strong pass rate (81%), while the proposed ReAct framework achieves a comparable AUP (0.705 ± 0.081) with 76% pass rate despite disabled internet access.

As shown in Appendix J.1, the performance gap relative to English contradicts the traditional resource-availability hierarchy. High-resource languages exhibit mixed results, while medium- and low-resource

**Table 4**
Cross-lingual performance on the ML$^2$B subset. $\mathrm{AUP}_{-1/\ln \tau}$ is reported as mean ; PR denotes Pass Rate (%). Higher is better. GPT-4.1-mini, GPT-OSS-120b, and Gemini-2.5-Flash are evaluated inside the AIDE execution. The proposed ReAct framework with GPT-OSS-120b is evaluated with the disabled internet access and external retrievers. In total, 2,520 runs are executed.

| **Language** | GPT-4.1-mini (AIDE) | Gemini-2.5-Flash (AIDE) | GPT-OSS-120b (AIDE) | GPT-OSS (ReAct) |
|---|---|---|---|---|
| | | $\mathrm{AUP}_{-1/\ln \tau}$ / PR | | |
| English | .756 ± .083 / 60 | .689 ± .157 / 84 | .600 ± .000 / 87 | .644 ± .063 / 78 |
| French | .711 ± .083 / 62 | .756 ± .175 / 80 | .556 ± .137 / 87 | .733 ± .144 / 73 |
| Spanish | .822 ± .083 / 51 | .689 ± .031 / 82 | .578 ± .083 / 91 | .733 ± .054 / 67 |
| Italian | .756 ± .137 / 62 | .800 ± .054 / 78 | .533 ± .054 / 96 | .622 ± .063 / 78 |
| Chinese | .711 ± .083 / 69 | .689 ± .166 / 84 | .667 ± .144 / 82 | .711 ± .031 / 73 |
| Japanese | .711 ± .191 / 71 | .778 ± .137 / 73 | .644 ± .063 / 73 | .667 ± .109 / 84 |
| Russian | .844 ± .083 / 56 | .800 ± .094 / 78 | .622 ± .063 / 78 | .711 ± .113 / 84 |
| Polish | .867 ± .054 / 49 | .711 ± .083 / 67 | .711 ± .083 / 78 | .778 ± .083 / 69 |
| Turkish | .822 ± .083 / 62 | .756 ± .113 / 73 | .689 ± .113 / 78 | .644 ± .083 / 84 |
| Arabic | .800 ± .109 / 60 | .689 ± .083 / 73 | .689 ± .083 / 73 | .800 ± .054 / 64 |
| Ukrainian | .622 ± .191 / 62 | .667 ± .000 / 80 | .711 ± .083 / 67 | .711 ± .083 / 80 |
| Romanian | .711 ± .113 / 64 | .756 ± .083 / 67 | .622 ± .083 / 78 | .711 ± .113 / 71 |
| Kazakh | .867 ± .094 / 67 | .711 ± .031 / 73 | .578 ± .083 / 89 | .711 ± .113 / 67 |
| Belarusian | .756 ± .175 / 60 | .756 ± .063 / 73 | .622 ± .083 / 78 | .689 ± .031 / 84 |
| **Avg.** | .768 ± .112 / 61 | .732 ± .091 / 76 | .630 ± .083 / 81 | .705 ± .081 / 76 |

languages display highly task-dependent behavior: certain competitions show better performance than English across multiple languages, while others exhibit substantial gaps. The variability within resource groups often exceeds the variability between them, suggesting that competition-specific characteristics (problem complexity, domain terminology, and code patterns) dominate over language resource availability. Task-characteristic analysis ($N$=15) further shows that feature count ($\rho$=+0.50) and domain-specific terminology ($\rho$=+0.44) are the strongest predictors of cross-lingual variance (Appendix J.3).

We identify five failure categories: execution errors, constraint violations, preprocessing inconsistencies, function signature modifications, and environment issues, with GPT-OSS more robust to these than GPT-4.1-mini (detailed breakdown and code examples in Appendix E). Our ReAct agent consumes on average 475K tokens per experiment with a consistently low output-to-input ratio (0.09); language and competition effects interact, so token efficiency cannot be attributed to either factor alone (Appendix K).

Taken together, our results reveal three consistent patterns. First, cross-lingual gaps are task-dependent rather than resource-dependent: within-group variability exceeds between-group variability across resource levels (Appendix J.1), and task characteristics, i.e., feature count ($\rho$=+0.50) and domain terminology ($\rho$=+0.44), predict cross-lingual variance better than language identity. Second, the primary bottleneck is pipeline instruction-following rather than language comprehension (Appendix E). Third, no model dominates on both quality and reliability: GPT-4.1-mini attains the highest AUP but the lowest pass rate, revealing a quality–robustness trade-off that aggregate scores obscure.

## 5. Conclusion

We introduce ML$^2$B, the first multilingual benchmark for evaluating LLMs on end-to-end ML pipeline generation, comprising 35 real Kaggle competitions translated into 14 languages (490 tasks across tabular, text, and image data), validated by native speakers. ML$^2$B integrates methodological advances for reproducible evaluation: private competitions to reduce data leakage, modular code submission to limit preprocessing leakage, and containerized execution, though modular submissions may retain residual data access if agents perform late-stage training, which remains a future direction. Our findings reveal that cross-lingual performance gaps are highly task-dependent, challenging the assumption that resource availability alone determines multilingual model capabilities.

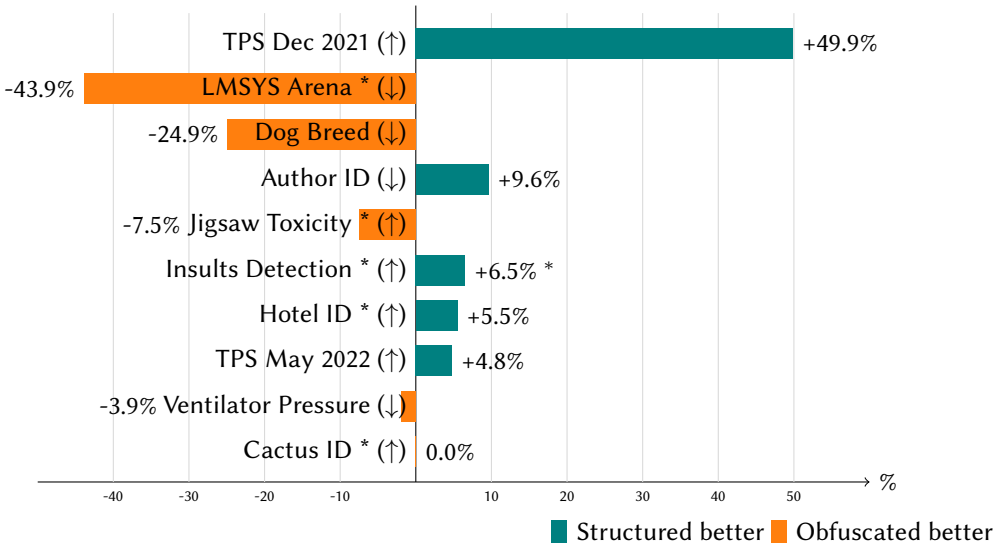



**Figure 4:** Relative improvement of ML$^2$B structured prompts over MLE-Bench obfuscated prompts on 10 shared competitions using DeepSeek-Reasoner within AIDE framework (3-hour runtime, 3 runs per configuration). Arrows denote metric direction (↑ higher, ↓ lower is better). Failed runs penalized with worst-case scores (0.0 for ↑, 100.0 for ↓). * marks the obfuscated format failures (2/3 pass rate).

## A. Prompt Format Comparison

We validate that structured prompt format aids model comprehension without enabling memorization-based shortcuts. Using DeepSeek-Reasoner within AIDE (3-hour runtime, 3 runs), we compare ML$^2$B structured prompts against MLE-Bench obfuscated prompts on 10 shared competitions. As shown in Figure 4, structured prompts achieve better performance on 5 of 10 competitions and demonstrate perfect stability (3/3 pass rate) across all tasks, whereas the obfuscated format fails (2/3 pass rate) on 5 competitions.

## B. Full Competition List

The full list of competitions is presented in Table 5.

## C. Domain Subjects Gathering

To assign a suitable domain to every competition in our benchmark we have decided on a following process. Firstly, the prompt containing the information about the problem (Figure 5) was given to the GPT-3.5-turbo model to derive domain tag for each competition. The next step was manual evaluation and necessary polishing of LLM-assigned domains in order to ensure their accuracy and suitability. Finally, 9 concise domains are obtained: media studies, social studies, environmental studies, urban studies, finance, marketing, healthcare and medical, manufacturing and quality, synthetic data. The number of competitions in each domain is presented in Figure 6.

## D. Prompt Templates

Prompts consist of three sections: (1) benchmark instructions, (2) competition-specific task description, and (3) submission constraints. Table 6 summarizes each section's content, while Table 7 shows the four supported function signature variants.

**Table 5**
The full list of competitions included in ML$^2$B

| comp name | Private/Public | Year | data type | Teams | Source |
|---|---|---|---|---|---|
| *WiDS Datathon 2020* | Public | 2020 | tabular | 951 | CODE4ML |
| *IEOR 242 Spring 2020 HW 4* | Public | 2020 | tabular | 72 | CODE4ML |
| *Explicit content detection* | Public | 2020 | text | 106 | CODE4ML |
| *109-1 NTUT Building Deep Learning Applications HW1* | Public | 2020 | image | 90 | CODE4ML |
| *UWaterloo STAT441/841 Data Challenge 1* | Public | 2020 | tabular | 131 | CODE4ML |
| *MADE HW-2* | Public | 2020 | text | 258 | CODE4ML |
| *Financial Engineering Competition (1/3)* | Public | 2020 | tabular | 290 | CODE4ML |
| *Financial Engineering Competition (2/3)* | Public | 2020 | tabular | 268 | CODE4ML |
| *Financial Engineering Competition (3/3)* | Public | 2020 | tabular | 268 | CODE4ML |
| *PRML-Data Contest-Nov 2020* | Public | 2020 – 2021 | tabular | 150 | CODE4ML |
| *Actuarial loss prediction* | Public | 2020 – 2021 | tabular | 140 | CODE4ML |
| *<She/Hacks> - Shaastra'21 and Wells Fargo* | Public | 2021 | tabular | 90 | CODE4ML |
| *ML2021Spring-hw1* | Public | 2021 | tabular | 2032 | CODE4ML |
| *2021-ML (Korean)* | Public | 2021 | tabular | 87 | CODE4ML |
| *SYDE 522 (Winter 2021)* | Public | 2021 | tabular | 130 | CODE4ML |
| *Tabular Playground Series - Jul 2021* | Public | 2021 | tabular | 1293 | CODE4ML |
| *Tabular Playground Series - Aug 2021* | Public | 2021 | tabular | 1753 | CODE4ML |
| *Classify Leaves* | Public | 2021 | image | 165 | CODE4ML |
| *Google Brain - Ventilator Pressure Prediction* | Public | 2021 | tabular | 2605 | CODE4ML |
| *Porto Seguro Data Challenge* | Public | 2021 | tabular | 174 | CODE4ML |
| *Crime Learn* | Public | 2021 | tabular | 96 | CODE4ML |
| *Binary Classification with a Tabular Stroke Prediction Dataset* | Public | 2023 | tabular | 770 | CODE4ML |
| *Multi-Class Prediction of Cirrhosis Outcomes* | Private | 2024 | tabular | 108 | CODE4ML |
| *Predicting Optimal Fertilizers* | Public | 2025 | tabular | 2648 | Kaggle |
| *Binary Prediction with a Rainfall Dataset* | Public | 2025 | tabular | 4381 | Kaggle |
| *Eid Al-Adha 2025: Sheep Classification Challenge* | Public | 2025 | image | 355 | Kaggle |
| *Alfa University income prediction* | Private | 2024 | tabular | 8 | Kaggle |
| *2024 DataLab Cup1* | Private | 2024 | text | 108 | Kaggle |
| *Thapar Summer School 2025 \| Hack-III* | Private | 2025 | tabular | 110 | Kaggle |
| *Rutgers Data101 Fall2022 Assignment 12* | Private | 2022 | tabular | 162 | Kaggle |
| *CS 506 Fall 2025 Technical Midterm* | Private | 2025 | tabular | 143 | Kaggle |
| *Car Becho Paisa Paao* | Private | 2025 | tabular | 302 | Kaggle |
| *ITMO Flat price prediction 2024* | Private | 2024 – 2025 | tabular | 127 | Kaggle |
| *Multi-label Classification Competition 2025* | Private | 2025 | image | 201 | Kaggle |
| *IFT6390-IFT3395: Beer Quality Prediction* | Private | 2025 | tabular | 192 | Kaggle |

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 5:** Example of the prompt used to derive competition domain

**Function Signature Variants.** We support four signature variants based on two dimensions as shown in Table 7. Full prompt templates with complete examples are available in our code repository.[1] **Monolithic** variant requires a single function:

```
def train_and_predict(X_train: dict, X_val: dict) -> np.ndarray:
    """Train model and return predictions."""
    ...
```

**Modular** variant separates training and inference:

[1] https://github.com/enaix/ml2b

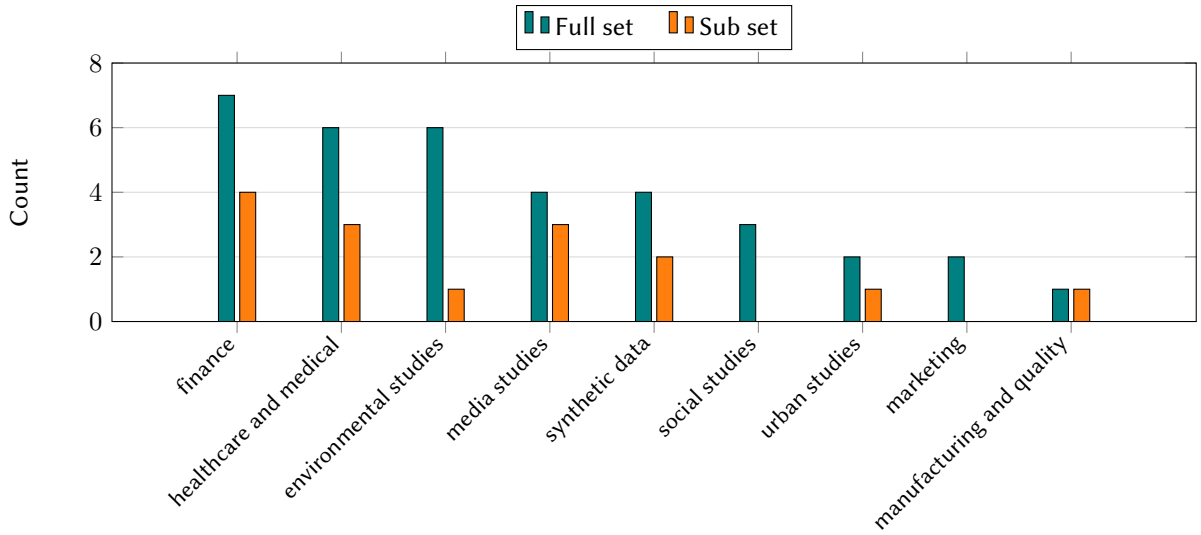



**Figure 6:** Domain distribution comparison on full and sub-set

**Table 6**
Prompt template structure.

| Section | Content |
|---|---|
| Benchmark instructions | Submission path requirements, train/test split clarification, programming language constraints, anti-plagiarism policy |
| Task description | ML task objective, domain, target metric, data schema with column descriptions (translated to target language) |
| Submission constraints | Required function signatures, single-file requirement, no global variables, self-contained code |

**Table 7**
Supported function signature variants.

| Type | Signature |
|---|---|
| Monolithic (extended) | `train_and_predict(...)` |
| Monolithic (TypedDict) | `train_and_predict(X_train, X_val)` |
| Modular (extended) | `train()`, `prepare_val()`, `predict()` |
| Modular (TypedDict) | Same with TypedDict |

```
def train(X_train: dict) -> Any: ...
def prepare_val(train_output: Any, X_val: dict) -> Any: ...
def predict(train_output: Any, val_output: Any) -> np.ndarray: ...
```

## E. Failure Pattern Analysis

Out of 2,520 submissions in the `MODULAR_PREDICT` format, <10% contain potential data leakage according to static analysis. In 81 submissions, leakage is detected in the `train` function, which does not operate on prediction data; in 33 cases it appears in trivial single-argument functions — both are false positives. This leaves 4% of submissions with potential actual leakage. Residual leakage may still occur if agents perform model training in later pipeline stages where both training and prediction data are theoretically accessible.

Table 8 categorizes the systematic failures observed across our experiments (pass rates across all languages and competitions are shown in Figure 7).

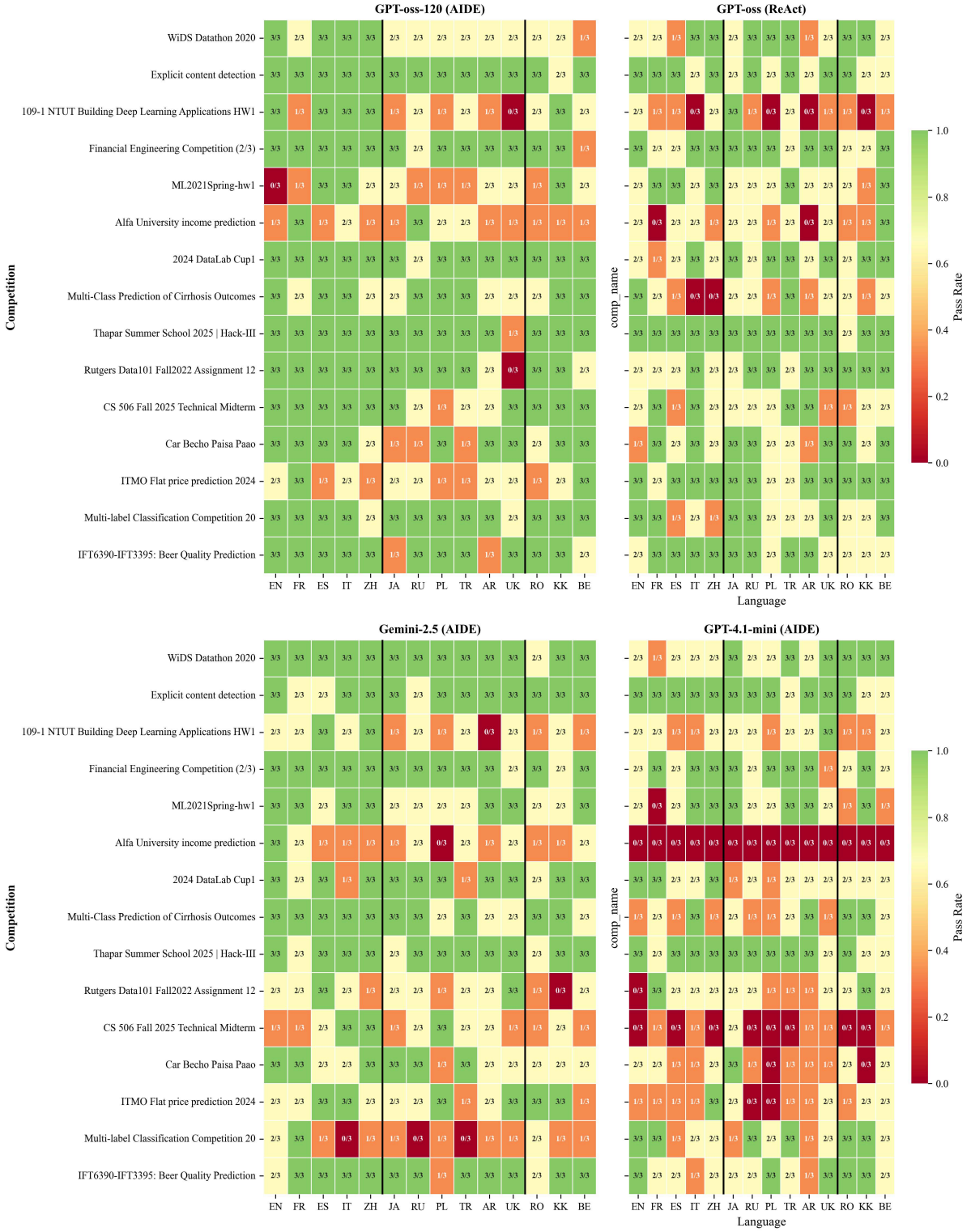



**Figure 7:** Pass rate across 14 languages and 15 competitions

Notably, these failures occurred more frequently with GPT-4.1-mini than with GPT-oss or Gemini-2.5-flash variants, suggesting that larger or more recent models may have better instruction-following capabilities for constrained code generation tasks.

Listings 1 and 2, 3 provide the examples of valid and two failed GPT-4.1-mini-generated codes, correspondingly.

**Table 8**
Detailed failure patterns observed across ML$^2$B experiments.

| Category | Description |
|---|---|
| Execution errors | Absence of `if __name__ == "__main__"` block prevented importing target functions from submission file |
| Data handling | Agents directly load data files within training functions instead of using function parameters, violating the predefined function signatures that expect data as arguments. |
| Preprocessing mismatch | Inconsistent feature engineering between training and validation sets, causing dimension or type errors |
| Signature violations | Modified function signatures despite explicit instructions requiring exact formats for grader compatibility |
| Global dependencies | Initialization code placed outside function definitions, violating self-contained code requirements |
| Environment issues | Use of deprecated API calls, non-existent library functions, or incorrect package versions |

Listing 1: Valid submission with proper modular structure and LightGBM pipeline.

```
import pandas as pd
import numpy as np
from typing import Any
from sklearn.model_selection import GroupKFold
from sklearn.metrics import mean_absolute_error
import lightgbm as lgb
def create_features(df):
    # Basic features
    df["R"] = df["R"].astype("category")
    df["C"] = df["C"].astype("category")
    # Lag features for u_in and u_out to capture time series
    df["u_in_lag1"] = df.groupby("breath_id")["u_in"].shift(1).fillna(0)
    df["u_in_lag2"] = df.groupby("breath_id")["u_in"].shift(2).fillna(0)
    df["u_out_lag1"] = df.groupby("breath_id")["u_out"].shift(1).fillna(0)
    df["u_out_lag2"] = df.groupby("breath_id")["u_out"].shift(2).fillna(0)
    # Time step as feature
    df["time_step"] = df["time_step"].astype(np.float32)
    # We can also add cumulative sum of u_in per breath to capture progression
    df["u_in_cumsum"] = df.groupby("breath_id")["u_in"].cumsum()
    # Drop columns not used for training
    return df
def train(X_train: pd.DataFrame, y_train: pd.DataFrame) -> Any:
    X_train = create_features(X_train)
    features = [
        "R",
        "C",
        "time_step",
        "u_in",
        "u_out",
        "u_in_lag1",
        "u_in_lag2",
        "u_out_lag1",
        "u_out_lag2",
        "u_in_cumsum",
    ]
    categorical_features = ["R", "C", "u_out", "u_out_lag1", "u_out_lag2"]
    # Prepare dataset for LightGBM
    lgb_train = lgb.Dataset(
        X_train[features],
        label=y_train["pressure"],
        categorical_feature=categorical_features,
        free_raw_data=False,
    )
    params = {
        "objective": "regression",
        "metric": "mae",
        "boosting_type": "gbdt",
        "learning_rate": 0.1,
        "num_leaves": 31,
        "max_depth": -1,
        "verbose": -1,
        "seed": 42,
        "feature_pre_filter": False,
    }
    model = lgb.train(params, lgb_train, num_boost_round=100)
    return (model, features, categorical_features)
def prepare_val(X_val: pd.DataFrame, train_output: Any) -> Any:
    model, features, categorical_features = train_output
    X_val = create_features(X_val)
    return X_val[features]
```

```
def predict(train_output: Any, prepare_val_output: Any) -> np.array:
    model, features, categorical_features = train_output
    X_val = prepare_val_output
    preds = model.predict(X_val, num_iteration=model.best_iteration)
    return preds
def run(X_train: pd.DataFrame, y_train: pd.DataFrame, X_val: pd.DataFrame) -> np.array:
    train_output = train(X_train, y_train)
    X_val_processed = prepare_val(X_val, train_output)
    preds = predict(train_output, X_val_processed)
    return preds
```

Listing 2: Failed submission illustrating API version incompatibility (line 55). The agent attempts to use `early_stopping_rounds` parameter in `lgb.train()`, which was deprecated in LightGBM 3.0.0 and removed in 4.0.0. This demonstrates a systematic limitation: agents do not verify library version compatibility or adapt their code when receiving error feedback about deprecated APIs.

```
import pandas as pd
import numpy as np
from typing import Any
from sklearn.model_selection import GroupKFold
from sklearn.metrics import mean_absolute_error
import lightgbm as lgb
import os
def feature_engineering(df: pd.DataFrame) -> pd.DataFrame:
    # Basic features
    df["u_in_cumsum"] = df.groupby("breath_id")["u_in"].cumsum()
    df["u_in_lag1"] = df.groupby("breath_id")["u_in"].shift(1).fillna(0)
    df["u_in_lag2"] = df.groupby("breath_id")["u_in"].shift(2).fillna(0)
    df["u_out_lag1"] = df.groupby("breath_id")["u_out"].shift(1).fillna(0)
    df["time_step_diff"] = df.groupby("breath_id")["time_step"].diff().fillna(0)
    # Interaction features
    df["R*C"] = df["R"] * df["C"]
    df["R*u_in"] = df["R"] * df["u_in"]
    df["C*u_in"] = df["C"] * df["u_in"]
    return df
def train(X_train: pd.DataFrame, y_train: pd.DataFrame) -> Any:
    X_train = feature_engineering(X_train)
    features = [
        "R",
        "C",
        "time_step",
        "u_in",
        "u_out",
        "u_in_cumsum",
        "u_in_lag1",
        "u_in_lag2",
        "u_out_lag1",
        "time_step_diff",
        "R*C",
        "R*u_in",
        "C*u_in",
    ]
    X_train = X_train[features]
    y = y_train["pressure"].values
    # GroupKFold on breath_id to avoid leakage
    # We will train a single model on full data here for simplicity
    params = {
        "objective": "regression",
        "metric": "mae",
        "boosting_type": "gbdt",
        "learning_rate": 0.1,
        "num_leaves": 64,
        "feature_fraction": 0.8,
        "bagging_fraction": 0.8,
        "bagging_freq": 5,
        "verbose": -1,
        "seed": 42,
        "n_jobs": -1,
    }
    lgb_train = lgb.Dataset(X_train, y)
    model = lgb.train(
        params,
        lgb_train,
        num_boost_round=1000,
        valid_sets=[lgb_train],
        early_stopping_rounds=50, # <-- Unsupported since version 4.0.0
        verbose_eval=100,
    )
    return (model, features)
def prepare_val(X_val: pd.DataFrame, train_output: Any) -> Any:
    model, features = train_output
    X_val = feature_engineering(X_val)
    X_val = X_val[features]
    return X_val
def predict(train_output: Any, prepare_val_output: Any) -> np.array:
    model, _ = train_output
    X_val = prepare_val_output
    preds = model.predict(X_val, num_iteration=model.best_iteration)
    return preds
def run(X_train: pd.DataFrame, y_train: pd.DataFrame, X_val: pd.DataFrame) -> np.array:
    train_output = train(X_train, y_train)
    prepared_val = prepare_val(X_val, train_output)
    preds = predict(train_output, prepared_val)
    return preds
```

Listing 3: Data leakage through target-dependent feature engineering. In train() (line 35), the agent concatenates input features with the target variable and applies add_features(), which generates lag features based on the target column pressure (lines 13–14). Reusing this feature engineering pipeline on the validation set (lines 62, 70), where the target is not available, results in a missing-column error.

```
1  import pandas as pd
2  import numpy as np
3  from typing import Any, Tuple
4  from sklearn.model_selection import train_test_split
5  from sklearn.metrics import mean_absolute_error
6  import lightgbm as lgb
7  def add_features(df: pd.DataFrame) -> pd.DataFrame: # <-- using target column presure
8      # Sort by breath_id and time_step for lag features
9      df = df.sort_values(["breath_id", "time_step"]).reset_index(drop=True)
10     # Lag features for u_in and pressure
11     df["u_in_lag1"] = df.groupby("breath_id")["u_in"].shift(1).fillna(0)
12     df["u_in_lag2"] = df.groupby("breath_id")["u_in"].shift(2).fillna(0)
13     df["pressure_lag1"] = df.groupby("breath_id")["pressure"].shift(1).fillna(0)
14     df["pressure_lag2"] = df.groupby("breath_id")["pressure"].shift(2).fillna(0)
15     # Rolling mean of u_in over last 3 steps
16     df["u_in_roll3"] = (
17         df.groupby("breath_id")["u_in"]
18         .rolling(3, min_periods=1)
19         .mean()
20         .reset_index(0, drop=True)
21     )
22     # Rolling std dev of u_in over last 3 steps
23     df["u_in_roll3_std"] = (
24         df.groupby("breath_id")["u_in"]
25         .rolling(3, min_periods=1)
26         .std()
27         .reset_index(0, drop=True)
28         .fillna(0)
29     )
30     # Time step squared to capture nonlinear trend
31     df["time_step2"] = df["time_step"] ** 2
32     return df
33 def train(X_train: pd.DataFrame, y_train: pd.DataFrame) -> Any:
34     # Add features to training data
35     X_train = add_features(pd.concat([X_train, y_train], axis=1))
36     # Target
37     y = X_train["pressure"].values
38     # Drop columns not used as features
39     drop_cols = ["id", "breath_id", "pressure"]
40     features = [c for c in X_train.columns if c not in drop_cols]
41     X = X_train[features]
42     # LightGBM dataset
43     lgb_train = lgb.Dataset(X, label=y)
44     params = {
45         "objective": "regression",
46         "metric": "mae",
47         "verbosity": -1,
48         "boosting_type": "gbdt",
49         "learning_rate": 0.1,
50         "num_leaves": 31,
51         "feature_fraction": 0.9,
52         "bagging_fraction": 0.9,
53         "bagging_freq": 1,
54         "seed": 42,
55         "n_jobs": -1,
56     }
57     model = lgb.train(params, lgb_train, num_boost_round=300)
58     return (model, features)
59 def prepare_val(X_val: pd.DataFrame, train_output: Any) -> Any:
60     model, features = train_output
61     # Add features to validation data (pressure not available, fill with 0)
62     X_val = add_features(X_val.copy()) # <-- pressure target column not presented
63     # For lags of pressure which we don't have in val, fill with 0
64     X_val["pressure"] = 0.0
65     X_val["pressure_lag1"] = 0.0
66     X_val["pressure_lag2"] = 0.0
67     X_val = add_features(X_val)
68     return X_val[features]
69 def predict(train_output: Any, prepare_val_output: Any) -> np.array:
70     model, features = train_output
71     X_val = prepare_val_output
72     preds = model.predict(X_val)
73     return preds
74 def run(X_train: pd.DataFrame, y_train: pd.DataFrame, X_val: pd.DataFrame) -> np.array:
75     train_output = train(X_train, y_train)
76     val_features = prepare_val(X_val, train_output)
77     preds = predict(train_output, val_features)
78     return preds
```

# F. 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.

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.

## G. Form Example

Figure 9 illustrates an example of one question block in a form, which asks a native speaker of Romanian to validate the translation of the competition description.

Translation Check

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

## H. Validation Results

As it has been stated before, for each language we considered the translations from a single annotator. To prove the reliability and the quality of translations, we have used back-translation into English and have assessed quality using BLEU metric.

Using the same GPT-4o model and identical prompts, we have executed three independent translation runs and collected the resulting English back-translations. We then have computed BLEU scores between the original texts in English and their back-translated versions. Finally, we have estimated bootstrap confidence intervals for each language and text type.

Lavie [40] claims that a BLEU score in the range [0.4, 0.5] can be a sign of high-quality translation. As shown in Figure 10, the mean BLEU scores across the three runs exceed 0.35 for all languages. Moreover, the intervals for descriptions and data cards are relatively narrow, suggesting stable and accurate translations.

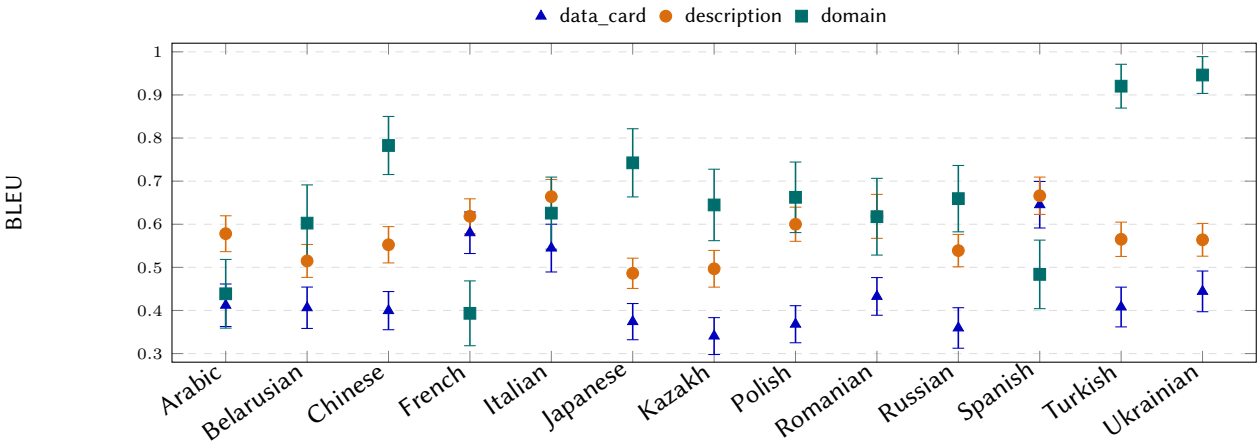



**Figure 10:** BLEU scores with 95% confidence intervals by language and text type.

For domain-level texts, the confidence intervals are notably wider. This is expected since BLEU is highly sensitive to short texts, and GPT-4o often produces less consistent translations for very short inputs lacking contextual cues. Overall, these results support our claim that translations from a single native speaker are sufficiently reliable for our study.

Table 9 shows the inter-annotator agreement measured by Gwet's AC1.

**Table 9**
Inter-annotator agreement measured by Gwet's AC1 for nativeness and meaning preservation criteria across languages.

| Language | Nativeness AC1 | Meaning AC1 |
|---|---|---|
| Arabic | 0.936 | 0.902 |
| Belarusian | 0.950 | 0.888 |
| Chinese | 0.450 | 0.971 |
| French | 0.723 | 0.899 |
| Italian | 0.841 | 0.971 |
| Japanese | 0.308 | 1.000 |
| Kazakh | 0.359 | 1.000 |
| Polish | 0.743 | 0.931 |
| Romanian | 0.560 | 0.869 |
| Russian | 0.876 | 0.971 |
| Spanish | 0.773 | 0.822 |
| Turkish | 0.876 | 1.000 |
| Ukrainian | 0.538 | 1.000 |

We also analyze the GPT-4 translation patterns. Prior work shows that GPT-4 produces more accurate and more lexically diverse translations than commercial systems such as Google Translate [34]. Additionally, Raunak et al. [41] note that GPT-family models tend to generate non-literal translations, including figurative renderings of idioms. Based on this, we anticipated many outputs that preserved meaning but lacked features that make them sound fully native.

Figure 11 indicates that nearly two-thirds of responses within each language are judged as both natural and semantically equivalent. The second most frequent pattern is NO and YES, reflecting translations that preserve meaning but lack native-like phrasing. This outcome is expected for LLMs, which tend to prioritize semantic adequacy over fine-grained stylistic and idiomatic conventions [42].

## I. AUP Metric Details

Here we provide the details on the modified versions of AUP metric.

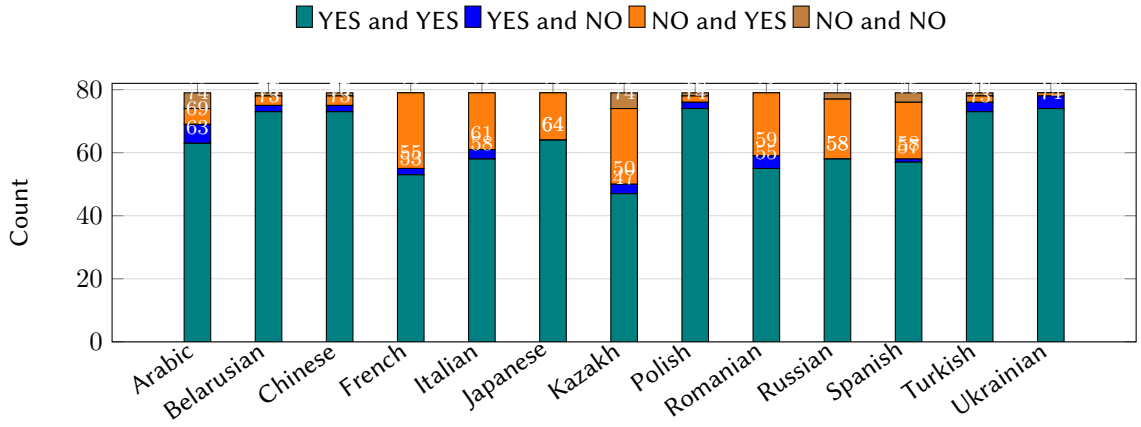



**Figure 11:** Answer distribution across languages

**Weighted AUP with Emphasis on Strong Outperformance.**

$$\mathrm{AUP}_{-\ln\tau} = \int_0^1 \rho(\tau) \cdot (-\ln\tau)\, d\tau \tag{5}$$

The weight $w(\tau) = -\ln\tau$ is strictly positive on $(0, 1)$, with $\int_0^1 (-\ln\tau) d\tau = 1$, ensuring natural normalization. This metric is sensitive to models achieving $r_p \ll 1$.

**Weighted AUP with Emphasis on Near-Baseline Performance.**

$$\mathrm{AUP}_{-1/\ln\tau} = \frac{\int_\epsilon^{1-\epsilon} \rho(\tau) \cdot \left(-\frac{1}{\ln\tau}\right) d\tau}{\int_\epsilon^{1-\epsilon} \left(-\frac{1}{\ln\tau}\right) d\tau} \tag{6}$$

where $\epsilon = 10^{-8}$. This metric emphasizes $\tau \approx 1$, distinguishing models that reliably meet baseline from those slightly below.

**Sensitivity Properties of Weighted AUP Metrics** The weighting functions transform the standard AUP into complementary metrics with distinct sensitivity profiles. For $\mathrm{AUP}_{-\ln\tau}$, the weight $w(\tau) = -\ln\tau$ diverges as $\tau \to 0^+$, making the metric highly sensitive to models achieving $r_{p,s} \ll 1$—i.e., models that substantially outperform the baseline. This metric, therefore, highlights **excellence**, emphasizing large performance gains and breakthrough capabilities.

Conversely, for $\mathrm{AUP}_{-1/\ln\tau}$, the weight $w(\tau) = -1/\ln\tau$ diverges as $\tau \to 1^-$, amplifying differences in the critical region where models are only marginally better than the baseline ($r_{p,s} \approx 1$). This metric emphasizes **reliability**, rewarding consistent, modest improvements over the baseline.

Together, these metrics provide a more nuanced diagnostic picture than the standard AUP alone. In multilingual evaluation, a model might show similar $\mathrm{AUP}_{\mathrm{std}}$ scores across languages, yet $\mathrm{AUP}_{-\ln\tau}$ could reveal that its advantage in high-resource languages comes from occasional strong wins , while $\mathrm{AUP}_{-1/\ln\tau}$ might indicate more stable, incremental improvements in others (Figure 12).

# J. Extended Analysis

## J.1. Cross-Lingual Performance Gap

Figure 13 shows the per-competition performance gap relative to English across 13 languages for GPT-OSS-120b.

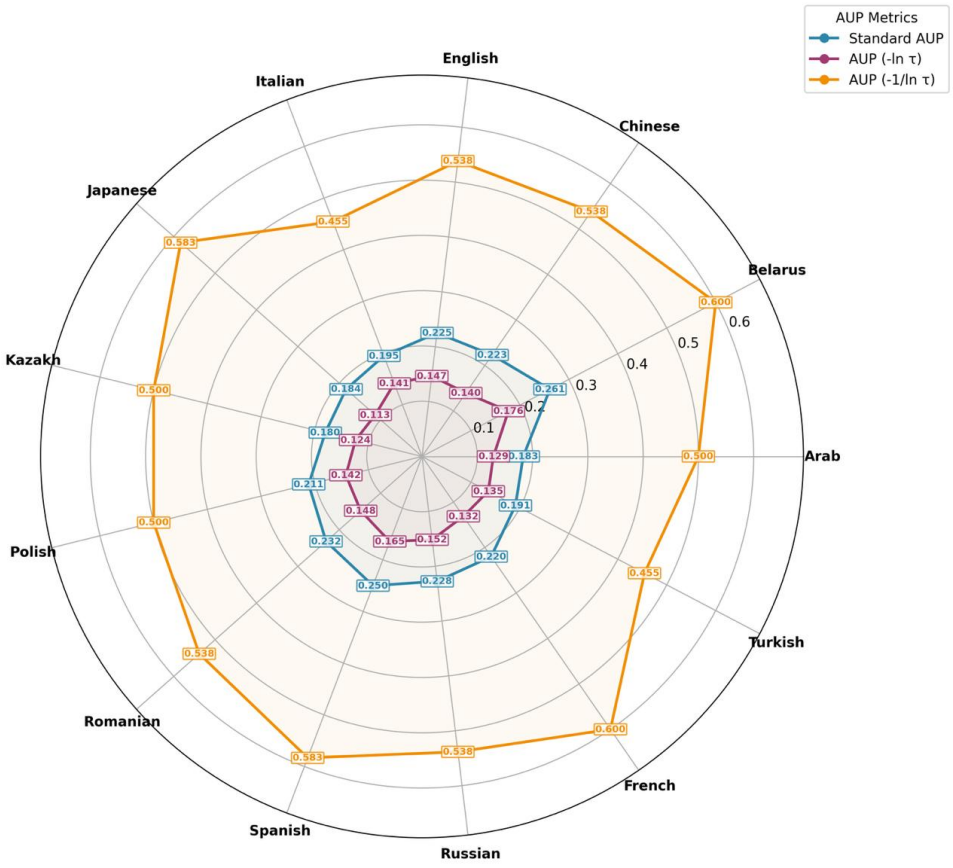



**Figure 12:** Aggregated performance across all languages for the three AUP variants.

## J.2. Qwen2.5-72B-Instruct Preliminary Evaluation

We run preliminary experiments with Qwen2.5-72B-Instruct via AIDE (1-hour budget) on 3 competitions $\times$ 3 languages (Russian, Kazakh, Chinese). Results are shown in Table 10.

**Table 10**
Qwen2.5-72B-Instruct preliminary results (F1 / RMSE / Logloss). None indicates a failed submission.

| Competition (metric) | Russian | Kazakh | Chinese |
|---|---|---|---|
| Explicit content (F1) | 0.941 | 0.971 | 0.967 |
| ML2021Spring-hw1 (RMSE) | None | None | None |
| ECE460j-fall24 (Logloss) | None | 0.495 | None |

Qwen fails to produce valid submissions in 5 of 9 pairs, with complete failure on the RMSE task. When Qwen succeeds — on the F1 task — scores are broadly comparable to GPT-OSS-120b (0.941–0.971 vs. 0.965–0.988). Failure analysis reveals a *configuration leakage* pattern: Qwen generates structurally correct `train`/`predict` functions but additionally emits global-scope code with hardcoded paths from its internal AIDE test run. When ML$^2$B executes only the required functions in a clean environment, stale path references cause runtime failures. This is consistent with the broader pattern that pipeline instruction-following is the primary bottleneck for weaker models. A full evaluation with matched 3-hour budgets across all 14 languages will be included in future work.


**Figure 13:** Performance gap relative to English across 13 languages for GPT-OSS-120b. Negative values indicate worse performance than English.

## J.3. Task-Characteristic Analysis of Cross-Lingual Variance

For each of the 15 competitions, we code domain, modality, dataset size, number of features, description length (tokens), and number of domain-specific terms. We compute per-competition cross-lingual standard deviation of performance ratios (std of $r_p$ across 14 languages, aggregated over all four model-framework configurations) and correlate these with task characteristics (Table 11).

**Table 11**
Spearman correlations between task characteristics and cross-lingual variance ($N$=15). Mean $r_p$ is uncorrelated with all characteristics (all $|\rho| < 0.17, p > 0.54$).

| Task characteristic | $\rho$ | p |
|---|---|---|
| Number of features | +0.50 | 0.055 |
| Domain-specific terms (count) | +0.44 | 0.100 |
| Dataset size (log) | +0.36 | 0.187 |
| Description length (tokens) | +0.04 | 0.874 |

Task characteristics predict *how much* a result varies across languages, not how well models perform on average. Low cross-lingual variance has two distinct patterns: tasks where all languages succeed (e.g., Thapar Summer School, std=0.0005) and tasks where all languages fail equally due to underspecification (e.g., Financial Engineering 2/3, std=0.034, mean $r_p \approx 0.60$). Stability is therefore not always a success signal. These correlations should be interpreted as exploratory evidence ($N$=15 limits statistical power) motivating targeted follow-up.

## K. Token Consumption Analysis

We analyze the token usage across 14 natural languages and 15 programming competitions using the proposed modification of ReAct agent, comprising a total of 630 individual experiments. Token usage is measured separately for input (prompt), output (generated code), and total tokens.

On average, experiments have consumed 475,158 total tokens (SD = 446,482) (for language detailed see Figure 14), including 448,755 input tokens (SD = 422,395) and 26,402 output tokens (SD = 42,534). The output-to-input token ratio remains consistently low across languages ($M = 0.09$, $SD = 0.02$), indicating substantial contextual prompts and concise code outputs.

**Language Effects** A one-way analysis of variance (ANOVA) revealed statistically detectable differences across languages for all token types:

- Input tokens: $F(13, 594) = 0.49$, $p < 0.001$
- Output tokens: $F(13, 594) = 0.68$, $p < 0.001$
- Total tokens: $F(13, 594) = 0.44$, $p < 0.001$

Romanian requires the highest number of tokens ($M = 584{,}788$), Polish the fewest ($M = 396{,}858$), representing a 47.4% difference (187,930 tokens per task). Post-hoc t-tests with Bonferroni correction ($\alpha = 0.000549$) identify no significant pairwise differences.

**Competition Effects** Competition type significantly affect token usage: $F(14, 593) = 4.09$, $p < 0.001$. The most token-intensive competition is rutgers-data101-fall2022-assignment-12 ($M = 686{,}101$), while financial-engineering-2 requires the fewest tokens ($M = 254{,}701$), corresponding to a 169.4% difference.

**Input-Output Correlation** Pearson correlation shows moderate positive association between input and output tokens ($r = 0.532$, $p < 0.001$). Language-specific correlations range from $r = 0.427$ (Kazakh) to $r = 0.851$ (Japanese).

**Interaction Effects** Within-language variation across competitions ($CV = 0.560$) exceeds between-language variation ($CV = 0.096$), indicating meaningful language $\times$ competition interactions.

Overall, the observed language $\times$ competition interaction suggests that token efficiency cannot be fully characterized by language or task alone (Figure 15), highlighting the importance of jointly modeling both dimensions when estimating the computational costs of large language models.

## Acknowledgments

We would like to thank Ivan Yamshchikov for proposing the initial idea of investigating how the natural language of prompts influences code generation.

Also, we would like to thank the translation assessors for their time and effort: Aldan Yerbalanov, Artyom Senokosov, Valeria Aitova, Charlene Madida, Cristian Buliga, Deng Yuying, Ekaterina Fedorishcheva, Eleonore Vissol-Gaudin, Eugene, Giorgio Vedovi, Mikhail Khvorostianyi, Michal Kucharzewski, O.L., Saraa Ali, Sahra, Zhang Yujing, Jinyu Zhou.

## Declaration on Generative AI

During the preparation of this work, the authors used Claude Sonnet 4.6 and Grammarly in order to: Grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

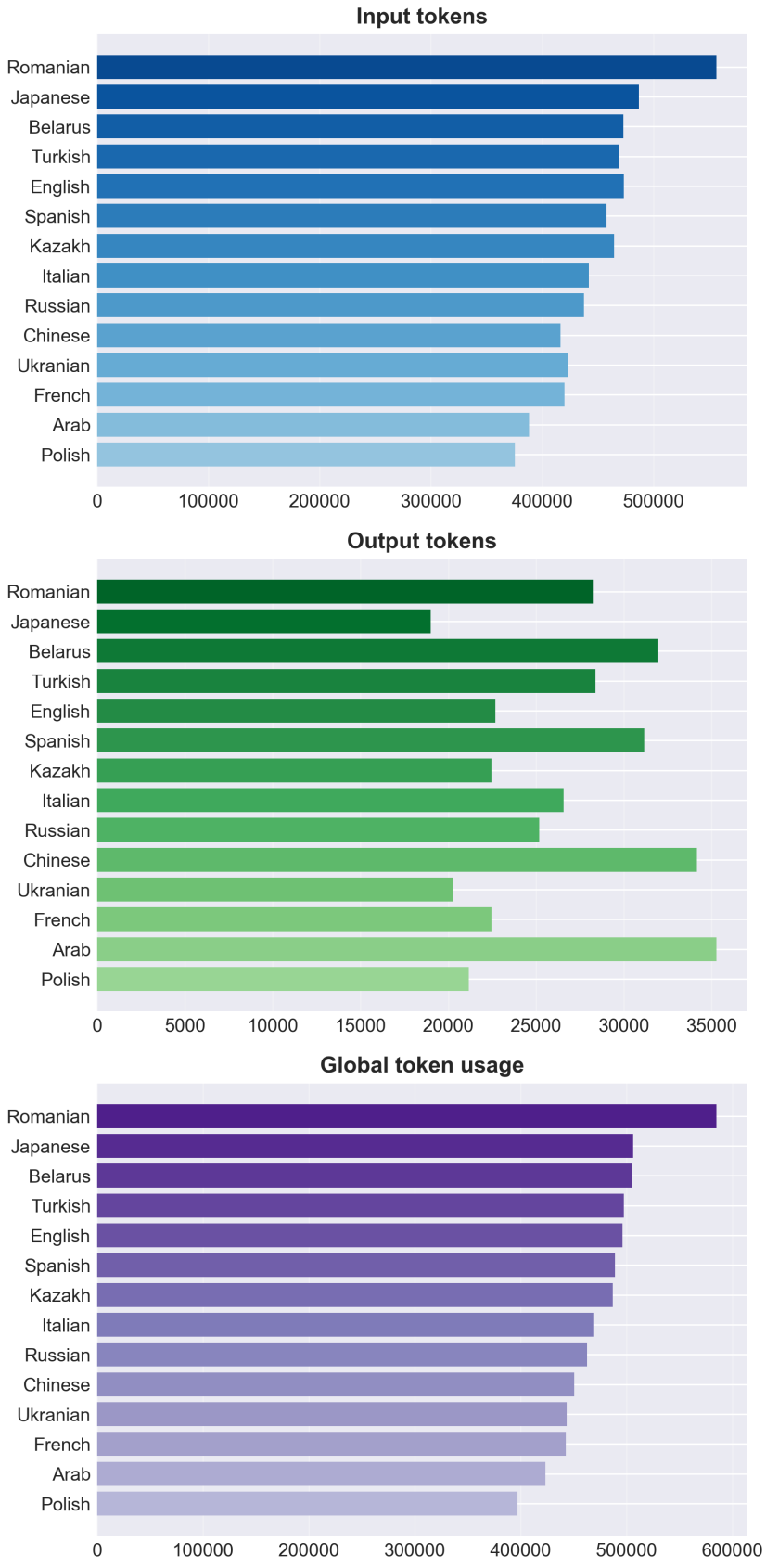



**Figure 14:** Mean token usage per language

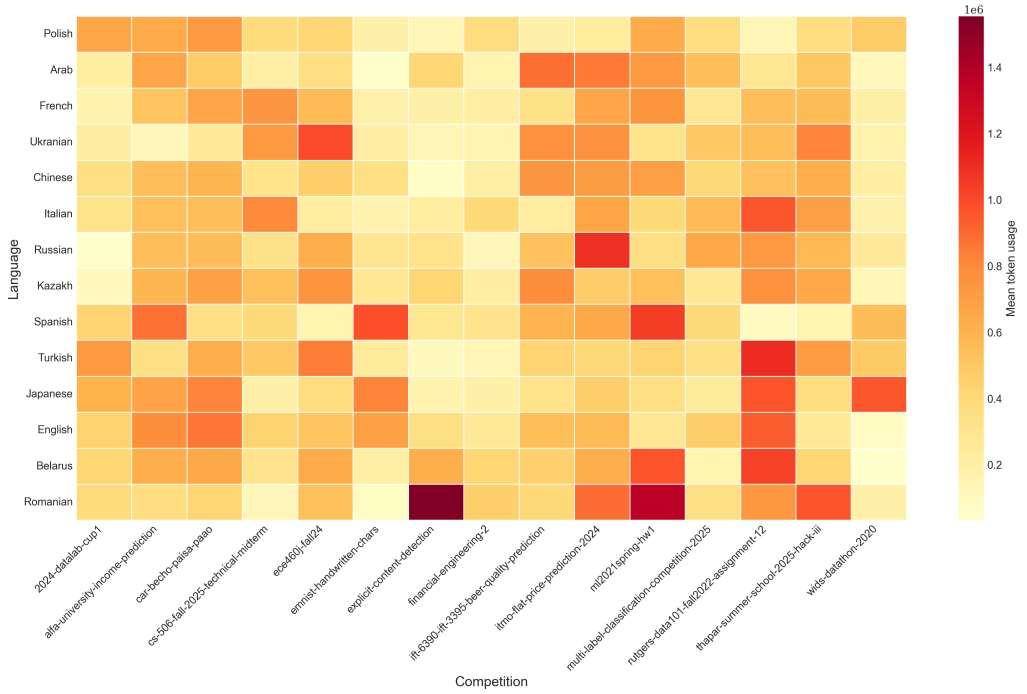



**Figure 15:** Language x Competition token consumption

## References


[1] M.-A. Zöller, M. F. Huber, Benchmark and survey of automated machine learning frameworks, J. Artif. Int. Res. 70 (2021) 409–472. URL: https://doi.org/10.1613/jair.1.11854. doi:10.1613/jair.1.11854.

[2] J. S. Chan, N. Chowdhury, O. Jaffe, J. Aung, D. Sherburn, E. Mays, G. Starace, K. Liu, L. Maksin, T. Patwardhan, A. Madry, L. 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.

[3] Y. Huang, J. Luo, Y. Yu, Y. Zhang, F. Lei, Y. Wei, S. He, L. Huang, X. Liu, J. Zhao, K. Liu, DA-code: Agent data science code generation benchmark for large language models, in: Y. Al-Onaizan, M. Bansal, Y.-N. Chen (Eds.), Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Miami, Florida, USA, 2024, pp. 13487–13521. URL: https://aclanthology.org/2024.emnlp-main.748/. doi:10.18653/v1/2024.emnlp-main.748.

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

[5] Z. Li, Y. Shi, Z. Liu, F. Yang, A. Payani, N. Liu, M. Du, Language ranker: A metric for quantifying llm performance across high and low-resource languages, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 28186–28194.

[6] L. Qin, Q. Chen, Y. Zhou, Z. Chen, Y. Li, L. Liao, M. Li, W. Che, P. S. Yu, A survey of multilingual large language models, Patterns 6 (2025).

[7] W. Xuan, R. Yang, H. Qi, Q. Zeng, Y. Xiao, Y. Xing, J. Wang, H. Li, X. Li, K. Yu, N. Liu, Q. Chen, D. Teodoro, E. Marrese-Taylor, S. Lu, Y. Iwasawa, Y. Matsuo, I. Li, Mmlu-prox: A multilingual benchmark for advanced large language model evaluation, ArXiv abs/2503.10497 (2025). doi:10.48550/arxiv.2503.10497.

[8] N. Raihan, A. Anastasopoulos, M. Zampieri, mHumanEval - a multilingual benchmark to evaluate large language models for code generation, in: L. Chiruzzo, A. Ritter, L. 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), Association for Computational Linguistics, Albuquerque, New Mexico, 2025, pp. 11432–11461. URL: https://aclanthology.org/2025.naacl-long.570/. doi:10.18653/v1/2025.naacl-long.570.

[9] A. Matton, T. Sherborne, D. Aumiller, E. Tommasone, M. Alizadeh, J. He, R. Ma, M. Voisin, E. Gilsenan-McMahon, M. Gallé, On leakage of code generation evaluation datasets, 2024. URL: https://arxiv.org/abs/2407.07565. arXiv:2407.07565.

[10] X. Zhou, M. Weyssow, R. Widyasari, T. Zhang, J. He, Y. Lyu, J. Chang, B. Zhang, D. Huang, D. Lo, Lessleak-bench: A first investigation of data leakage in llms across 83 software engineering benchmarks, 2025. URL: https://arxiv.org/abs/2502.06215. arXiv:2502.06215.

[11] A. Apicella, F. Isgrò, R. Prevete, Don't push the button! exploring data leakage risks in machine learning and transfer learning, Artificial Intelligence Review 58 (2025). URL: http://dx.doi.org/10.1007/s10462-025-11326-3. doi:10.1007/s10462-025-11326-3.

[12] C. Yang, R. A. Brower-Sinning, G. A. Lewis, C. Kästner, Data leakage in notebooks: Static detection and better processes, 2022. URL: https://arxiv.org/abs/2209.03345. arXiv:2209.03345.

[13] 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, K. R. Patil, Overview of leakage scenarios in supervised machine learning, Journal of Big Data 12 (2025). URL: http://dx.doi.org/10.1186/s40537-025-01193-8. doi:10.1186/s40537-025-01193-8.

[14] S. Kapoor, A. Narayanan, Leakage and the reproducibility crisis in machine-learning-based science, Patterns 4 (2023) 100804. URL: https://www.sciencedirect.com/science/article/pii/S2666389923001599. doi:https://doi.org/10.1016/j.patter.2023.100804.

[15] S. Ouyang, D. Huang, J. Guo, Z. Sun, Q. Zhu, J. M. Zhang, Dscodebench: A realistic benchmark for data science code generation, arXiv preprint arXiv:2505.15621 (2025).

[16] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, T. 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.

[17] H. Padigela, C. Shah, D. Juyal, Ml-dev-bench: Comparative analysis of ai agents on ml development workflows, 2025. URL: https://arxiv.org/abs/2502.00964. arXiv:2502.00964.

[18] Q. Huang, J. Vora, P. Liang, J. Leskovec, Mlagentbench: Evaluating language agents on machine learning experimentation, 2024. URL: https://arxiv.org/abs/2310.03302. arXiv:2310.03302.

[19] A. Vakhrushev, A. Ryzhkov, M. Savchenko, D. Simakov, R. Damdinov, A. Tuzhilin, Lightautoml: Automl solution for a large financial services ecosystem, 2022. URL: https://arxiv.org/abs/2109.01528. arXiv:2109.01528.

[20] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, G. Neubig, Openhands: An open platform for ai software developers as generalist agents, 2025. URL: https://arxiv.org/abs/2407.16741. arXiv:2407.16741.

[21] Z. Liu, Y. Cai, X. Zhu, Y. Zheng, R. Chen, Y. Wen, Y. Wang, W. E, S. Chen, Ml-master: Towards ai-for-ai via integration of exploration and reasoning, 2025. URL: https://arxiv.org/abs/2506.16499. arXiv:2506.16499.

[22] X. Yang, X. Yang, S. Fang, Y. Zhang, J. Wang, B. Xian, Q. Li, J. Li, M. Xu, Y. Li, H. Pan, Y. Zhang, W. Liu, Y. Shen, W. Chen, J. Bian, R&d-agent: An llm-agent framework towards autonomous data science, 2025. URL: https://arxiv.org/abs/2505.14738. arXiv:2505.14738.

[23] Z. Wang, G. Cuenca, S. Zhou, F. F. Xu, G. Neubig, Mconala: A benchmark for code generation from multiple natural languages, arXiv preprint arXiv:2203.08388 (2022).

[24] A. Cosma, I.-B. Iordache, P. Rosso, RoCode: A dataset for measuring code intelligence from problem definitions in Romanian, in: N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, N. Xue (Eds.), Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), ELRA and ICCL, Torino, Italia, 2024, pp. 14173–14185. URL: https://aclanthology.org/2024.lrec-main.1236/.

[25] M. Li, A. Mishra, U. 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).
[26] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, Q. V. Do, Y. Xu, P. Fung, A multitask, multilingual, multimodal evaluation of ChatGPT on reasoning, hallucination, and interactivity, in: J. C. Park, Y. Arase, B. Hu, W. Lu, D. Wijaya, A. Purwarianti, A. A. 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), Association for Computational Linguistics, Nusa Dua, Bali, 2023, pp. 675–718. URL: https://aclanthology.org/2023.ijcnlp-main.45/. doi:10.18653/v1/2023.ijcnlp-main.45.
[27] K. Ahuja, H. Diddee, R. Hada, M. Ochieng, K. Ramesh, P. Jain, A. Nambi, T. Ganu, S. Segal, M. Ahmed, K. Bali, S. 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.
[28] N. Muennighoff, T. Wang, L. Sutawika, A. Roberts, S. Biderman, T. Le Scao, M. S. Bari, S. Shen, Z. X. Yong, H. Schoelkopf, X. Tang, D. Radev, A. F. Aji, K. Almubarak, S. Albanie, Z. Alyafeai, A. Webson, E. Raff, C. Raffel, Crosslingual generalization through multitask finetuning, in: A. Rogers, J. Boyd-Graber, N. Okazaki (Eds.), Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Toronto, Canada, 2023, pp. 15991–16111. URL: https://aclanthology.org/2023.acl-long.891/. doi:10.18653/v1/2023.acl-long.891.
[29] M. B. Moumoula, A. K. Kabore, J. Klein, T. F. Bissyande, Evaluating programming language confusion, arXiv preprint arXiv:2503.13620 (2025).
[30] W. Huo, X. Feng, Y. Huang, C. Fu, B. Li, Y. Ye, Z. Zhang, D. Tu, D. Tang, Y. Lu, et al., Enhancing non-english capabilities of english-centric large language models through deep supervision fine-tuning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 24185–24193.
[31] U. Shaham, J. Herzig, R. Aharoni, I. Szpektor, R. Tsarfaty, M. Eyal, Multilingual instruction tuning with just a pinch of multilinguality, arXiv preprint arXiv:2401.01854 (2024).
[32] E. Trofimova, E. Sataev, A. Drozdova, P. Guseva, A. Scherbakova, A. Ustyuzhanin, A. Gorodilova, V. Berezovskiy, Code4ml 2.0: a large-scale dataset of annotated machine learning code, 2024. URL: https://doi.org/10.5281/zenodo.12700065. doi:10.5281/zenodo.12700065.
[33] A. Drozdova, E. Trofimova, P. Guseva, A. Scherbakova, A. Ustyuzhanin, Code4ml: a large-scale dataset of annotated machine learning code, PeerJ Computer Science 9 (2023) e1230.
[34] W. Jiao, W. Wang, J. tse Huang, X. Wang, S. Shi, Z. Tu, Is chatgpt a good translator? yes with gpt-4 as the engine, 2023. URL: https://arxiv.org/abs/2301.08745. arXiv:2301.08745.
[35] K. Gwet, Computing inter-rater reliability and its variance in the presence of high agreement., The British journal of mathematical and statistical psychology 61 Pt 1 (2008) 29–48. doi:10.1348/000711006x126600.
[36] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, A. Smola, Autogluon-tabular: Robust and accurate automl for structured data, arXiv preprint arXiv:2003.06505 (2020).
[37] Z. Tang, H. Fang, S. Zhou, T. Yang, Z. Zhong, T. Hu, K. Kirchhoff, G. Karypis, Autogluon-multimodal (automm): Supercharging multimodal automl with foundation models, arXiv preprint arXiv:2404.16233 (2024).
[38] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, 2004. URL: https://arxiv.org/abs/cs/0102001. arXiv:cs/0102001.
[39] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: The eleventh international conference on learning representations, 2022.
[40] A. Lavie, Evaluating the output of machine translation systems, in: Proceedings of the 9th Conference of the Association for Machine Translation in the Americas: Tutorials, Association for Machine Translation in the Americas, Denver, Colorado, USA, 2010. URL: https://aclanthology.org/2010.amta-tutorials.4/.

[41] V. Raunak, A. Menezes, M. Post, H. H. Awadalla, Do gpts produce less literal translations?, 2023. URL: https://arxiv.org/abs/2305.16806. arXiv:2305.16806.

[42] K. Mahowald, A. A. Ivanova, I. Blank, N. Kanwisher, J. B. Tenenbaum, E. Fedorenko, Dissociating language and thought in large language models, Trends in Cognitive Sciences 28 (2024) 517–540. doi:10.1016/j.tics.2024.01.011.