JOR-Bench: Japanese Operations Research Benchmarks for Large Language Models

Yuu Jinnai
CyberAgent
Tokyo, Japan
jinnai_yu@cyberagent.co.jp

arXiv:2607.16777v1 [cs.CL] 18 Jul 2026

Abstract

We present JOR-Bench, a collection of five Japanese-language benchmarks for evaluating the ability of large language models (LLMs) to formulate and solve operations research (OR) problems. Each benchmark is a Japanese translation of an existing English benchmark: IndustryOR, MAMO Complex LP, NL4OPT, OptiBench, and OptMATH, covering 1,319 problems spanning linear programming, mixed-integer programming, non-linear programming, and combinatorial optimization. JOR-Bench is a solver-independent benchmark that can be used with any solver or programming language, and consists of pairs of Japanese problem statements and expected numerical answers. We evaluate seven LLMs, including multilingual general-purpose models and Japanese-specialized models, on both the original English and the new Japanese versions, and compare performance across languages. For the main evaluation, we standardize execution with the Python interface to OR-Tools to make model outputs comparable and reproducible with open-source software. Our results show that OR formulation ability is largely language-neutral for strong multilingual models; the overall average accuracy difference between English and Japanese is only -0.3 pp. Yet error analysis reveals subtle cross-lingual differences, including a pragmatic disambiguation failure in some domains that causes models to output decision-variable values instead of the objective value when the prompt is in Japanese.

1 Introduction

Operations research (OR) is concerned with mathematically optimizing decisions in complex systems (Hillier and Lieberman, 2005). Formulating a real-world problem as a mathematical program is a cognitively demanding task that has traditionally required specialized expertise. With the rapid improvement of large language models (LLMs) in mathematical and code-generation tasks, there is growing interest in whether LLMs can automate OR problem formulation (Ramamonjison et al., 2023).

Existing benchmarks for evaluating LLM-based OR formulation are predominantly English-language resources (Yang et al., 2025; Lu et al., 2025; Huang et al., 2025a). This creates a gap for languages other than English: practitioners who work in non-English environments cannot reliably know whether an LLM will handle their natively phrased problems as well as English-phrased equivalents. It is also of interest to the research community to understand whether LLMs' mathematical formulation ability is language-neutral or whether there are systematic cross-lingual differences.

In this paper we introduce JOR-Bench, a collection of five Japanese-language benchmarks for evaluating LLMs' OR formulation ability derived from translating existing English benchmarks. We conduct a cross-lingual evaluation of seven LLMs, including multilingual general-purpose models and Japanese-specialized models, on both the original English and the new Japanese versions of the benchmarks. We find that overall OR formulation ability is largely language-neutral for strong multilingual models, with an average accuracy difference of only -0.3 pp between English and Japanese.

2 JOR-Bench: Japanese OR Benchmarks

2.1 Source Benchmarks

JOR-Bench translates five existing English OR benchmarks into Japanese.

IndustryOR (Huang et al., 2025a) contains 100 real-world industrial optimization problems drawn from manufacturing, logistics, and supply chain management. Problems typically involve multi-period planning with several decision variables and realistic constraints.

MAMO Complex LP (Huang et al., 2025b) consists of 203 linear programming problems collected

and reviewed through a combination of human annotation and LLM-assisted quality checks. All problems can be solved with a standard LP solver.

NL4OPT (Ramamonjison et al., 2023) is derived from the NL4OPT competition held in 2022. The dataset contains 245 small LP problems expressed in English. Problems typically involve two to three decision variables and a handful of linear constraints.

OptiBench (Yang et al., 2025) covers a wide variety of OR problems, including nonlinear programming problems gathered from textbooks by human workers.

OptMATH (Lu et al., 2025) is a synthetic dataset targeting hard combinatorial optimization and MIP problems, including job-shop scheduling, facility dispersion, and aircraft landing scheduling.

2.2 Translation Methodology

We translate all five English datasets into Japanese using LLM-based translation followed by human post-editing. Because many source problems contain Markdown tables, structured data, and mathematical notation, preserving the input format is as important as translating the natural-language text. We use gpt-oss-120b for translation because, in preliminary checks, it preserved tables and mixed-format problem statements more reliably than standard machine translation models we tested. The first author then post-edited the translations to check terminology, numerical consistency, and faithfulness to the original English problem statements.

2.3 Evaluation Protocol

Table 1 shows an example entry from JOR-Bench. Each entry pairs a Japanese problem description with the expected numerical answer. The benchmark is therefore solver-independent: it can evaluate any LLM-generated solution as long as the final answer can be compared numerically.

For the experiments in this paper, we use the following execution-based protocol. We prompt each model to generate a standalone Python script that formulates and solves the given optimization problem (see Appendix B). The model is required to surround the Python code with a fenced code block (triple backticks), which is a standard Markdown feature. If such a code block is not found, we mark the problem as a "no-code" failure. If a code block is found, we extract the code and attempt to execute it in a sandboxed Python environment with OR-Tools installed. The sandbox enforces a maximum execution time of 300 seconds per problem and a 32GB memory limit. If execution exceeds either limit, we mark the problem as a "timeout" or "memory limit exceeded" failure. If the output of the run matches the expected answer within a tolerance of 10^{-6}, we mark the problem as correct; otherwise, it is marked as incorrect. Models occasionally generate code that uses only the Python standard library without importing OR-Tools. We allow this behavior as long as the output is correct, because the benchmark is intended to measure problem-solving accuracy rather than strict compliance with a particular solver API.

3 Experiments

The main experiments use OR-Tools as the execution backend. To evaluate backend sensitivity, we additionally report MAMO results on PuLP and Pyomo in Appendix A.

3.1 Models

We evaluate seven LLMs spanning a range of sizes, training objectives, and language specializations. All the models are reasoning models. Table 2 provides a summary. Swallow is a Japanese-specialized model family continually pretrained from Qwen-3 series weights (Fujii et al., 2024; Ma et al., 2025) to improve the Japanese language capabilities. We use the fine-tuned versions of Swallow as they demonstrate math and coding tasks roughly on par with Qwen-3 series models. While Swallow is Japanese-specialized, it is trained to reason in English. CAT-Thinking (Jinnai, 2026) is a Japanese-specialized model further fine-tuned from Swallow-8B. CAT-Thinking is trained on a wide range of reasoning tasks including mathematical formulation and code generation and forced to generate reasoning trace in Japanese. Thus, at least on surface level, no English interferes the generation of the output when CAT-Thinking is prompted in Japanese. The performance of CAT-Thinking in the standard math and coding benchmarks is slightly lower than Swallow-8B. Using these models allows us to evaluate the effect of Japanese specialization on OR formulation ability in two languages. We additionally evaluate Nemotron, a Japanese-specialized model trained on a mixture of Japanese and English data (NVIDIA, 2025), and GPT-OSS, a multilingual general-purpose model with a high competency (**?**).

Japanese	English	Answer
ある会計事務所は、パートタイム従業員とフルタイム従業員を雇用しています。フルタイム従業員は1シフトにつき8時間働き、パートタイム従業員は1シフトにつき4時間働きます。さらに、フルタイム従業員はシフトごとに300ドル、パートタイム従業員はシフトごとに100ドルを支払われます。現在、会計事務所には500時間の労働時間を必要とするプロジェクトがあります。予算が15,000ドルの場合、合計労働者数を最小にするために、各タイプの労働者を何人ずつスケジュールすべきでしょうか。	An accounting firm employs part time workers and full time workers. Full time workers work 8 hours per shift while part time workers work 4 hours per shift. In addition, full time workers are paid $300 per shift while part time workers are paid $100 per shift. Currently, the accounting firm has a project requiring 500 hours of labor. If the firm has a budget of $15000, how many of each type of worker should be scheduled to minimize the total number of workers.	100

Table 1: Example entry from JOR-Bench.

Model	Full name	Params
Qwen3-8B	Qwen/Qwen3-8B	8B
Qwen3-32B	Qwen/Qwen3-32B	32B
Swallow-8B	tokyotech-llm/Qwen3-Swallow-8B-RL-v0.2 (Ma et al., 2025)	8B
Swallow-32B	tokyotech-llm/Qwen3-Swallow-32B-RL-v0.2 (Ma et al., 2025)	32B
CAT-Thinking	cyberagent/CAT-Thinking-8B (Jinnai, 2026)	8B
Nemotron	nvidia/NVIDIA-Nemotron-Nano-9B-v2-JP (NVIDIA, 2025)	9B
GPT-OSS	openai/gpt-oss-120b	120B

Table 2: Models evaluated. All seven models are run on both the English and Japanese versions of all five benchmarks.

Model	Indus	MAMO	NL4OPT	OptiB	OptM
CAT-Thinking	77.0	**85.2**	**81.6**	**77.9**	64.0
GPT-OSS	**81.0**	74.9	17.6	42.0	**73.2**
Qwen3-8B	35.0	38.9	76.7	57.9	10.4
Qwen3-32B	53.0	50.2	80.0	61.8	25.9
Swallow-8B	27.0	12.3	45.7	33.1	7.3
Swallow-32B	48.0	28.6	43.3	40.7	42.0
Nemotron	9.0	4.9	4.1	3.0	7.0

Table 3: Accuracy (%) on **English** benchmarks. Bold = best; underline = second best per column.

Model	Indus	MAMO	NL4OPT	OptiB	OptM
CAT-Thinking	**81.8**	**85.0**	71.0	**70.7**	66.3
GPT-OSS	**81.8**	73.5	21.2	46.4	**71.7**
Qwen3-8B	48.5	34.5	**78.4**	62.0	13.3
Qwen3-32B	55.6	50.0	70.6	58.8	19.5
Swallow-8B	23.2	10.5	46.5	31.7	7.2
Swallow-32B	50.5	37.5	42.9	41.7	45.1
Nemotron	3.0	1.0	4.5	4.1	1.2

Table 4: Accuracy (%) on **Japanese** benchmarks. Bold = best; underline = second best per column.

Model	Indus	MAMO	NL4	OptiB	OptM	Avg
CAT-Thinking	+4.8	−0.2	-10.6^{***}	-7.1^{**}	+2.2	−2.2
GPT-OSS	+0.8	−1.4	+3.7	$+4.5^{**}$	−1.5	+1.2
Qwen3-8B	$+13.5^{*}$	−4.4	+1.6	+4.1	+2.9	+3.5
Qwen3-32B	+2.6	−0.2	-9.4^{**}	−3.0	−6.4	−3.3
Swallow-8B	−3.8	−1.8	+0.8	−1.3	−0.1	−1.2
Swallow-32B	+2.5	$+8.9^{*}$	−0.4	+1.0	+3.2	+3.0
Nemotron	−6.0	−3.9	+0.4	+1.2	-5.8^{*}	−2.8
Avg	+2.1	−0.4	−2.0	−0.1	−0.8	−0.2

Table 5: Δ_{JA} (pp) = Japanese accuracy minus English accuracy. Significance from McNemar's test (continuity-corrected): $^{*}p<0.05$, $^{**}p<0.01$, $^{***}p<0.001$. Positive = Japanese is better; negative = English is better.

3.2 Results

Tables 3 and 4 show accuracy (%) on English and Japanese benchmarks, respectively.

4 Quantitative and Qualitative Analysis

Table 5 shows $\Delta_{JA} = \text{acc}_{JA} - \text{acc}_{EN}$ for all seven models across the five benchmarks.

Overall language neutrality. The grand average Δ_{JA} is -0.3 pp—statistically indistinguishable from zero. Most individual cells lie within ± 5 pp. This suggests that OR formulation ability, when assessed as code-generation accuracy, is largely language-agnostic for models with sufficient multilingual training.

Statistical significance. We apply McNemar's test (continuity-corrected) to per-problem binary outcomes for each model–dataset pair. Of 35 pairs, only 7 reach $p < 0.05$ (marked in Table 5). The four significant regressions concentrate on NL4OPT and OptiBench; the three significant improvements span IndustryOR (Qwen3-8B, $p = 0.026$), MAMO (Swallow-32B, $p = 0.030$), and OptiBench (GPT-OSS, $p = 0.008$). The remaining 28 cells are statistically indistinguishable

from zero.

Model scale interacts with language. Scale helps, especially on hard problems, regardless of language. Increasing model size within a family consistently improves accuracy, and the gains are largest on the hardest benchmark. On OptMATH (EN), Qwen3-32B more than doubles Qwen3-8B (25.9% vs. 10.4%) and Swallow-32B improves over Swallow-8B nearly six-fold (42.0% vs. 7.3%). Combinatorial optimization of this complexity appears to require substantial model capacity regardless of language specialization.

That being said, the two model families for which we evaluate two sizes exhibit *opposite* scaling behavior with respect to language. For Qwen3, the smaller 8B model improves in Japanese ($+3.5$ pp on average, the best of any model) while the larger 32B model regresses (-3.3 pp), with a -9.4 pp drop on NL4OPT and -6.4 pp on OptMATH. For Swallow, the pattern reverses: the 8B model declines in Japanese (-1.2 pp) while the 32B model improves ($+3.2$ pp), including a $+8.9$ pp gain on MAMO. These crossovers show that cross-lingual transfer is not a fixed property of a model family but depends on the interaction between scale and language specialization.

MAMO declines for most models. MAMO shows the most consistent direction of negative transfer: six of the seven models decline in Japanese, although the average drop is small (-0.4 pp) because Swallow-32B improves by $+8.9$ pp. We investigate this pattern in Section 4.1.

4.1 MAMO: Japanese Decline for Most Models

We analyze error types on MAMO to understand why most models perform slightly worse in Japanese. Table 6 breaks down failures into our error taxonomy. Backend sensitivity results with PuLP and Pyomo are reported separately in Appendix A; this subsection focuses on OR-Tools-specific error behavior.

Two patterns are observed from the error breakdown.

Import errors are language-independent. The dominant error type is "import error," caused by models attempting to use the deprecated `ortools.graph.pywrapgraph` API. This error occurs at nearly identical rates in English and Japanese for all models (within 1 pp), confirming that it reflects a knowledge gap about current API versions rather than a language effect.

Model	Lang	Correct	No-code	Import err
GPT-OSS	EN	74.9%	1.5%	18.7%
	JA	73.5%	**4.0%**	18.5%
Qwen3-8B	EN	38.9%	20.7%	31.0%
	JA	34.5%	**23.0%**	30.5%
Swallow-8B	EN	12.3%	37.9%	18.7%
	JA	10.5%	**47.5%**	14.0%

Table 6: Error breakdown on MAMO for selected models. "No-code" = no fenced Python block found; "Import err" = deprecated OR-Tools API call. Bold highlights the increase in "No-code" errors from EN to JA.

No-code errors increase in Japanese. A consistent finding across the three models that decline is that *no-code failures increase in Japanese*. Swallow-8B shows the most dramatic shift: 37.9% in English versus 47.5% in Japanese. This indicates that models are less likely to follow the implicit instruction to respond with a Python code block when the prompt is in Japanese. This is consistent with the broader finding in multilingual LLM research that instruction-following behavior degrades in non-primary languages (Dussolle et al., 2025).

4.2 Pragmatic Disambiguation Failures

We examine why CAT-Thinking regresses substantially on NL4OPT (-10.6 pp) and OptiBench (-7.1 pp) in Japanese.

NL4OPT error breakdown. In English, CAT-Thinking's primary failure mode is "wrong answer" (37/245 problems, 15.1%). In Japanese, wrong-answer failures increase to 56/245 (22.9%), accounting for the bulk of the regression. Runtime errors also increase slightly ($8 \rightarrow 13$).

A qualitative example illustrates the mechanism:

> *Problem (Japanese)*: An oil spill requires transporting ducks to shore. A boat carries 10 ducks per trip (20 min/trip); a canoe carries 8 ducks (40 min/trip). The boat may make at most 12 trips; at least 60% of all trips must be by canoe. At least 300 ducks must be transported. ボートとカヌーをそれぞれ何回使うべきでしょうか？
> *Expected answer*: 1160.0 (total time in minutes)

Model output: `12 23` (boat trips and canoe trips)

The model correctly solves the LP (12 boat trips, 23 canoe trips), but the Japanese question asks 「何回使うべきか」 ("how many times should each be used"), which it answers literally with decision variable values. The English version of the same problem more strongly implies reporting the optimal objective value (total time). This is a language-specific disambiguation failure: in English, the implicit pragmatics of "minimize" questions direct models to report the objective; in Japanese, the explicit phrasing leads the model to a different answer.

OptiBench error breakdown. Similarly, OptiBench wrong-answer errors increase from 51 (8.4%) in English to 88 (14.5%) in Japanese, while runtime errors remain approximately constant. This supports the hypothesis that CAT-Thinking's domain adaptation encodes stronger English-language pragmatics about what value to report as the answer.

Qwen3-32B: the same mechanism at larger scale. The same pragmatic disambiguation failure explains the significant Qwen3-32B regression on NL4OPT (-9.4 pp, $p = 0.005$). Wrong-answer errors more than double in Japanese: 9.8% (EN) $\rightarrow$ 18.4% (JA), while Qwen3-8B's wrong-answer rate is flat at 4.1% in both languages. The trend holds on OptiBench: Qwen3-32B wrong-answer errors rise from 6.9% to 13.2% in Japanese, while Qwen3-8B stays below 1.3%. These findings unify the two significant NL4OPT regressions: both CAT-Thinking (-10.6 pp, $p<0.001$) and Qwen3-32B (-9.4 pp, $p=0.005$) are driven by wrong-answer inflation in Japanese, whereas the smaller Qwen3-8B is unaffected. Larger models, despite higher overall accuracy, appear more prone to treating an explicit literal question (何回使うべきか, "how many times should each be used") as the primary signal for what value to output.

4.3 GPT-OSS on NL4OPT: A Format Mismatch

GPT-OSS scores only 17.6% (EN) and 21.2% (JA) on NL4OPT despite leading on OptMATH. The anomaly is language-independent: 78% of failures in both languages produce decision-variable output (e.g., `12 23` for boat and canoe trips) rather than the expected objective value (`1160.0`). Completions average $\sim$1,500 characters with no code truncation, ruling out token-limit exhaustion as an explanation.

Per-problem EN/JA agreement is high ($\kappa = 0.70$), driven by 76% of problems failing consistently in both languages—the same problems, the same failure mode. A plausible explanation is a training-distribution mismatch: NL4OPT problems frequently ask "how many of each type should be used," phrasing that may have been answered with variable counts in a distribution the model was fine-tuned on. The prompt explicitly requests the objective value (see Appendix B), but this instruction is overridden. Unlike the failures in Section 4.2, this is a language-independent benchmark-format issue.

4.4 Per-Problem EN/JA Agreement

Aggregate accuracy differences can conceal per-problem churn if a model solves *different* problems correctly in each language. We compute Cohen's κ between per-problem binary correctness in English and Japanese for each model–dataset pair.

The most striking result is Swallow-32B on OptMATH ($\kappa = -0.07$): despite a near-neutral aggregate difference (+3.2 pp), only 19% of problems are correct in *both* languages; 30% are correct only in Japanese and 23% only in English. Japanese and English thus act as complementary lenses on hard combinatorial problems, unlocking different subsets for this model. Qwen3-8B on OptMATH shows the same anti-correlated pattern ($\kappa = -0.07$, EN+JA+ $= 0.6\%$): at the edge of model capability, language choice switches solution and failure near-randomly. Conversely, GPT-OSS on NL4OPT achieves high agreement ($\kappa = 0.70$) for the wrong reason: 76% of problems are consistently wrong in both languages (Section 4.3). These contrasts show that κ reveals qualitatively different regimes hidden beneath similar-looking accuracy numbers.

4.5 Japanese-Specialized Models and Scale

Among the Japanese-specialized models, scale is decisive. Swallow-32B substantially outperforms Swallow-8B in Japanese on every benchmark (e.g., MAMO 37.5% vs. 10.5%, OptMATH 45.1% vs. 7.2%, OptiBench 41.7% vs. 31.7%), and unlike its smaller sibling it *improves* in Japanese relative to English (+3.2 pp on average). Nemotron, the other Japanese-specialized model, remains weak at every scale of comparison. This indicates that Japanese specialization alone is not sufficient for OR formulation: it must be paired with enough

Dataset	EN 8B	EN 32B	JA 8B	JA 32B
IndustryOR	−8.0	−5.0	−25.3	−5.1
MAMO	−26.6	−21.6	−24.0	−12.5
NL4OPT	−31.0	−36.7	−31.9	−27.7
OptiBench	−24.8	−21.1	−30.3	−17.1
OptMATH	−3.1	+16.1	−6.1	+25.6
Avg	−18.7	−13.7	−23.5	−7.4

Table 7: Family gap (pp) computed as Swallow accuracy minus Qwen3 accuracy at matched parameter scales. Positive values favor Swallow.

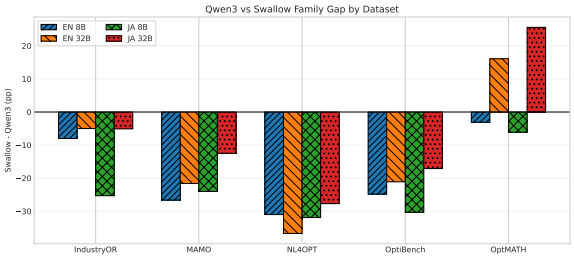

Figure 1: Qwen3 vs. Swallow family gaps across datasets and languages. Hatches and color choices are designed to remain distinguishable in grayscale and for color-vision deficiencies.

model capacity to reason about the optimization problem.

4.6 Qwen3 vs. Swallow: Effects of Japanese Continued Pretraining

Because Swallow models are built on Qwen3 with additional Japanese-focused continued pretraining, we compare the two families directly at matched scales. Table 7 reports the per-dataset accuracy gap (Swallow minus Qwen3, in percentage points).

Three findings are notable. First, Qwen3 is consistently stronger on LP-heavy datasets (IndustryOR, MAMO, NL4OPT, OptiBench) at both scales and in both languages. Second, Swallow-32B substantially outperforms Qwen3-32B on OptMATH (+16.1 pp in English and +25.6 pp in Japanese), indicating that Japanese continued pretraining can help on hard combinatorial optimization at larger scale. Third, the Japanese language setting narrows the average family gap at 32B (−13.7 to −7.4 pp), while widening it at 8B (−18.7 to −23.5 pp), showing that the effect of language-focused continued pretraining depends on model scale.

This family-level pattern is also backend-sensitive on MAMO (Appendix A): under PuLP, Swallow outperforms Qwen3 at both scales, while under Pyomo only Swallow-32B remains ahead.

Dataset	EN avg	JA avg	Δ
IndustryOR	47.1%	49.2%	+2.1
MAMO	42.1%	41.7%	−0.4
NL4OPT	49.9%	47.9%	−2.0
OptiBench	45.2%	45.1%	−0.1
OptMATH	32.8%	32.0%	−0.8

Table 8: Average accuracy across all seven models per dataset.

4.7 Dataset-Level Difficulty Analysis

Table 8 summarizes the average accuracy across models for each dataset and language.

Averaged across models, English and Japanese accuracy are close on every dataset (all $|\Delta| \leq 2$ pp), supporting the overall language-neutrality finding. OptMATH consistently shows the lowest average accuracy (32.8% EN, 32.0% JA), confirming it is the most challenging benchmark. The remaining datasets cluster between 42% and 50%, indicating comparable aggregate difficulty despite their different OR problem types.

5 Conclusion

We introduced JOR-Bench, five Japanese-language OR benchmarks translated from established English originals, together with a systematic evaluation of seven LLMs across both language versions.

Our main finding is that OR formulation ability is largely language-neutral: the grand average Δ_{JA} is only -0.3 pp across all model–dataset pairs. However, this aggregate masks important model-level and dataset-level patterns. McNemar's tests confirm that only 7 of 35 model–dataset pairs differ significantly; the remainder are statistically indistinguishable from zero. A shared mechanism explains the two prominent NL4OPT regressions: both CAT-Thinking (-10.6 pp, $p<0.001$) and Qwen3-32B (-9.4 pp, $p=0.005$) show wrong-answer errors doubling in Japanese, as Japanese phrasing elicits decision-variable output rather than the objective value, while Qwen3-8B at 8B scale is unaffected. GPT-OSS exhibits the same output-format issue on NL4OPT in *both* languages—a language-independent training-distribution mismatch. Most models also decline on MAMO in Japanese not due to reasoning failures but because they generate Python code blocks less reliably, a behavioral instruction-following gap that only the largest Japanese-specialized model (Swallow-32B) overcomes. Additional MAMO robustness experiments with PuLP and Pyomo (Appendix A) show

that backend choice strongly affects absolute accuracy, while the English–Japanese gap remains comparatively small. Per-problem agreement (Cohen's κ) further reveals that on hard combinatorial problems (OptMATH), Japanese and English unlock complementary problem subsets: Swallow-32B solves 30% of problems only in Japanese and 23% only in English, while the aggregate difference is just $+3.2$ pp.

These results underscore the value of Japanese OR benchmarks: English performance alone does not predict Japanese performance, and different models exhibit qualitatively different cross-lingual behaviors. We release all data and evaluation code to facilitate multilingual OR research.

6 Limitations

This study has several limitations. First, JOR-Bench is a translation benchmark rather than a benchmark written originally in Japanese. It therefore may not fully capture the discourse patterns, terminology, and implicit assumptions that appear in naturally occurring Japanese business or engineering optimization problems. Therefore, *the benchmark is Japanese-language but not culturally or contextually Japanese*. Second, our error taxonomy is coarse; a more fine-grained categorization (e.g., distinguishing constraint formulation errors from objective function errors) would provide deeper insight. Third, our execution-based evaluation is limited to Python solver interfaces (OR-Tools in the main experiments, with PuLP and Pyomo in the MAMO robustness study). Preliminary experiments with MiniZinc and JuMP produced too few executable formulations for most models to support a meaningful cross-lingual comparison; broader multi-language solver evaluation is left for future work.

References

Antoine Dussolle, Andrea Cardeña Díaz, Shota Sato, and Peter Devine. 2025. M-IFEval: Multilingual instruction-following evaluation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6176–6191, Albuquerque, New Mexico. Association for Computational Linguistics.

Kazuki Fujii, Taishi Nakamura, Mengsay Loem, Hiroki Iida, Masanari Ohi, Kakeru Hattori, Hirai Shota, Sakae Mizuki, Rio Yokota, and Naoaki Okazaki. 2024. Continual pre-training for cross-lingual LLM adaptation: Enhancing japanese language capabilities. In *First Conference on Language Modeling*.

Frederick S. Hillier and Gerald J. Lieberman. 2005. *Introduction to operations research*. McGraw-Hill College.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2025a. Orlm: A customizable framework in training large models for automated optimization modeling.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2025b. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2678–2710, Albuquerque, New Mexico. Association for Computational Linguistics.

Yuu Jinnai. 2026. Cost of reasoning in non-english languages: A case study on japanese. *arXiv preprint arXiv:2607.10114*.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. 2025. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Forty-second International Conference on Machine Learning*.

Youmi Ma, Sakae Mizuki, Kazuki Fujii, Taishi Nakamura, Masanari Ohi, Hinari Shimada, Taihei Shiotani, Koshiro Saito, Koki Maeda, Kakeru Hattori, Takumi Okamoto, Shigeki Ishida, Rio Yokota, Hiroya Takamura, and Naoaki Okazaki. 2025. Building instruction-tuning datasets from human-written instructions with open-weight large language models. In *Second Conference on Language Modeling*.

NVIDIA. 2025. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model. *arXiv preprint arXiv:2508.14444*.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and 1 others. 2023. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pages 189–203. PMLR.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. 2025. Optibench meets resocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*.

A Solver Backend Robustness on MAMO

Table 9 reports MAMO accuracy for additional execution backends using the same 203 problems and five models. Compared with OR-Tools, these PuLP and Pyomo runs show substantially lower

Model	PuLP EN	PuLP JA	Pyomo EN	Pyomo JA
GPT-OSS	70.4	73.9	64.5	59.1
Qwen3-8B	0.0	0.0	3.0	3.9
Qwen3-32B	2.0	2.0	1.5	2.5
Swallow-8B	13.8	11.8	0.0	0.5
Swallow-32B	31.5	27.1	12.3	5.9

Table 9: MAMO accuracy (%) by backend and language for five models with available PuLP and Pyomo runs (all out of 203 problems).

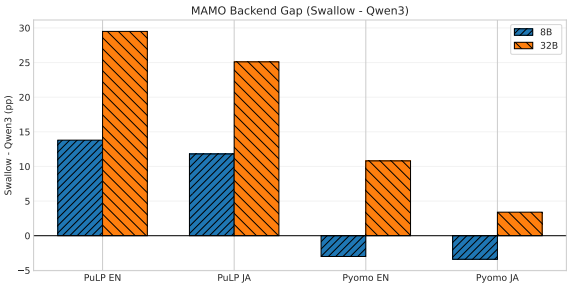

Figure 2: MAMO backend sensitivity for the Swallow–Qwen3 family gap. The figure highlights that backend choice changes the absolute gap pattern, even when the model family and dataset are fixed.

absolute accuracy overall. Averaged over the five models, accuracy is 23.5% (EN) and 23.0% (JA) for PuLP, and 16.3% (EN) and 14.4% (JA) for Pyomo. Thus, backend choice materially changes absolute performance, whereas the average cross-lingual gap remains relatively small (-0.6 pp for PuLP and -1.9 pp for Pyomo, JA minus EN).

B Prompts

The prompts used for the experiments are provided below. We use the same prompt template for all models and datasets, with only the dataset-specific problem description, library name, and task language changing.

> Please develop a Python script that uses [[Library Name]] library to solve the mathematical problem provided below.
>
> ## Instructions
>
> 1. Chain-of-Thought Before providing the code, explain the solution approach in [[Task Language]]. Your explanation should detail how to model the problem with [[Library Name]], specifically identifying the variables, constraints, and objective functions involved. 2. Python Script Implementation Following the [[Task Language]] explanation, provide the Python script. Ensure the implementation adheres to these specific guidelines:
>
> - Code Quality: Write clean, high-quality code. Include helpful comments to ensure the logic is accessible to beginners. - Formatting: Enclose the script within a triple backtick code fence so it can be easily parsed. - Exact Output: The script's output must contain only the final numerical value or answer. Do not include any descriptive text (e.g., "Answer: 42") or additional formatting. - Mandatory [[Library Name]] Usage: You must solve the problem using [[Library Name]]. Do not hardcode the final answer, even for simple problems. - Standalone Execution: The script must be self-contained and ready to run.
>
> ## Problem to Solve
>
> [[Problem Statement Here]]

Library	Version
Python	3.11.4
transformers	4.57.6
vllm	0.17.1
OR-Tools	9.10.4067
PuLP	2.4.0
Pyomo	6.4.0

Table 10: Libraries used for the experiments.

C Reproducibility Statement

The experiments are conducted using the following software libraries and versions, as summarized in Table 10. All the dataset used in the study are publicly available to facilitate reproducibility. The experiments are run on NVIDIA A100 GPUs with 40GB or 80GB memory.