Multilingual Code Explanation for Mainframe Languages

Kaoru Shinkawa
IBM Research
IBM Japan
Tokyo, Japan
kaoruma@jp.ibm.com

Ai Ishida
IBM Research
IBM Japan
Tokyo, Japan
aiishida@jp.ibm.com

Yasuharu Katsuno
IBM Research
IBM Japan
Tokyo, Japan
katsuno@jp.ibm.com

Fumiko Satoh
IBM Research
IBM Japan
Tokyo, Japan
sfumiko@jp.ibm.com

***Abstract*—Mainframe systems, written in legacy languages such as COBOL, PL/I, and JCL, continue to support mission-critical applications across various industries. Their complexity and limited documentation hinder maintenance and modernization, especially in regions where localized explanations are essential for accurate understanding. However, existing approaches predominantly generate English-only outputs and rely on resource-intensive models unsuitable for secure, on-premises environments. This study explores multilingual explanation generation for mainframe programs using lightweight language models suitable for constrained enterprise settings. We evaluate two strategies—(a) direct generation in the target language and (b) translation-based generation from English—across five languages: Japanese, French, German, Spanish, and Portuguese. Explanation quality is assessed using BLEU, ROUGE-L, METEOR, and semantic similarity. Preliminary results show that lightweight models can produce semantically adequate multilingual explanations. Translation-based generation generally yields higher lexical and structural quality across languages and models, while direct generation shows promise in specific scenarios. These findings demonstrate the feasibility of deploying multilingual explanation systems in enterprise environments and highlight opportunities to refine generation strategies based on language and code characteristics.**

***Index Terms*—Mainframe Systems, Multilingual Code Explanation, Legacy Programming Languages, Application Modernization**

I. Introduction

Mainframe systems built with legacy languages such as COBOL, PL/I, and JCL continue to support critical enterprise applications worldwide. Over decades, these systems have evolved into complex codebases deeply embedded in organizational infrastructure, making them difficult to maintain and modernize due to limited documentation and intricate logic. Understanding the underlying code is essential for modernization efforts, and code explanation helps make program logic more accessible and actionable [1].

This need is especially pronounced in non-English-speaking regions, where engineers require explanations in their native languages to avoid misinterpretation. However, most existing tools focus solely on English-language outputs, limiting accessibility for global teams [2]. Explanation quality varies depending on both the programming and native languages involved. Benchmarks such as HumanEval-XL [3] show that performance in code-related tasks fluctuates across language combinations, underscoring the importance of cross-lingual generalization.

Mainframe applications often operate in secure, on-premises environments, making large-scale language models impractical due to their computational demands and reliance on cloud infrastructure. This creates a need for lightweight models that can be deployed locally while still delivering high-quality multilingual explanations.

To address these challenges, we investigate multilingual code explanation for legacy mainframe programs using lightweight language models that are suitable for deployment in constrained enterprise environments. Our study targets five native languages (Japanese, French, German, Spanish, and Portuguese) and three legacy programming languages (COBOL, PL/I, and JCL), evaluating two strategies: direct generation in the target language and translation-based generation from English. Explanation quality is assessed using lexical and semantic metrics to determine the feasibility and trade-offs of each approach.

Our contributions are threefold: (1) we propose a multilingual explanation framework tailored for legacy mainframe code using lightweight models; (2) we compare direct and translation-based generation strategies across five native languages and three programming languages; and (3) we provide insights into explanation quality and model behavior under enterprise deployment constraints.

In the following section, we review existing approaches to code explanation and identify gaps related to multilingual support and deployment feasibility, which motivate the design of our experimental framework.

II. Related Works

Recent research on code explanation has primarily focused on leveraging large language models (LLMs) to enhance software comprehension, particularly for legacy systems such as COBOL, PL/I, and JCL [1]. These models have demonstrated strong performance in generating English-language summaries [2], but their reliance on cloud-based infrastructure and substantial computational resources limits their applicability in secure, on-premises enterprise environments.

Moreover, most existing approaches lack robust multilingual capabilities, despite the growing need for localized explana-

2151-0849/25/$31.00 ©2025 IEEE
DOI 10.1109/ASEW67777.2025.00014

tions in non-English-speaking regions. This gap is especially critical for legacy systems, which are still widely used in global industries.

Instruction-tuned models like CodeT5+ and CodeLlama have shown promising results in generating coherent summaries for modern programming languages such as Java, Python, and C# [4], [5]. However, legacy languages remain underrepresented in multilingual code understanding research. Studies such as Prather et al. [6] suggest that prompting in native languages (e.g., Arabic, Chinese, Portuguese) can improve code generation outcomes, though terminology mismatches pose challenges.

Additionally, recent work has highlighted the potential of lightweight models to perform competitively in code generation tasks, offering a practical balance between performance and efficiency [7]. These findings raise the question of whether similar benefits can be achieved in multilingual code explanation, particularly for legacy systems in constrained environments.

Building on these insights, our study investigates the use of lightweight models for multilingual explanation of legacy mainframe code, addressing both deployment constraints and language accessibility. These considerations inform the design of our experimental framework, which we present in the following section along with our research questions and evaluation methodology.

III. Methods

A. Research Questions

To evaluate the feasibility and effectiveness of lightweight language models for multilingual explanation of mainframe code, we address the following research questions:

RQ1: Can lightweight language models generate high-quality multilingual explanations for legacy mainframe code? We evaluate whether models suitable for on-premises deployment can produce accurate and semantically faithful explanations across multiple native languages.

RQ2: Which strategy is more effective: direct generation in the target language or translation-based generation from English? We compare two approaches—generating explanations directly in each language versus translating English explanations—to assess trade-offs in quality and consistency.

RQ3: Are explanation quality and generation behavior consistent across different programming languages and target native languages? We analyze whether certain languages exhibit systematic differences in explanation quality or semantic fidelity, and whether these differences are model-dependent.

B. Experiment Setup

To explore the feasibility of multilingual code explanation for mainframe programs in resource-constrained enterprise environments, we designed experiments targeting five native languages—Japanese, French, German, Spanish, and Portuguese—and three legacy programming languages—COBOL, PL/I, and JCL.

TABLE I
Program Samples for the experiments

Program Language	Program Statistics		
	Number of Samples	*Total Length*[a]	*Average Length*[a]
COBOL	30	12423	414.1
PL/I	22	2037	92.6
JCL	18	187	10.4

[a]Length of each code represents the number of lines of code (LOC).

We investigate two strategies for generating multilingual explanations (Fig. 1):

(a) **Direct Generation**: Explanations are generated directly in each target language.

(b) **Translation-Based Generation**: Explanations are first generated in English and then translated into the target languages.

We adopted a zero-shot prompting approach without any task-specific fine-tuning. Each prompt was designed to elicit structured and informative responses, focusing on three key aspects of the program:

- Business purpose: What the program is intended to achieve from a business or operational perspective.
- Inputs and outputs: The data consumed and produced by the program.
- Detailed functional summary: Detailed breakdown of the program's logic and operations.

Prompts were slightly adapted for each programming language to account for syntax and structural differences. For direct generation, the prompt included an instruction to produce the explanation in the target native language (e.g., "Please explain the following program in Japanese."). For translation-based generation, the explanation was first generated in English, followed by a translation instruction (e.g., "Please translate the following English text into Japanese.").

These strategies were evaluated in terms of explanation quality, semantic fidelity, and suitability for deployment in on-premises environments with limited computational resources.

C. Dataset

To ensure consistency and comparability across experiments, we curated a benchmark consisting of basic functional program samples from real-world COBOL, PL/I, and JCL programs (Table. I). These samples were chosen from English-based applications to reflect common programming constructs and typical operational procedures found in legacy systems, while avoiding overly complex or domain-specific logic. To standardize inputs, all comments were removed during preprocessing, enabling evaluation of the models' ability to generate and translate explanations based solely on code semantics. Reference explanations were initially generated using a large-scale model and subsequently verified and refined by human annotators.

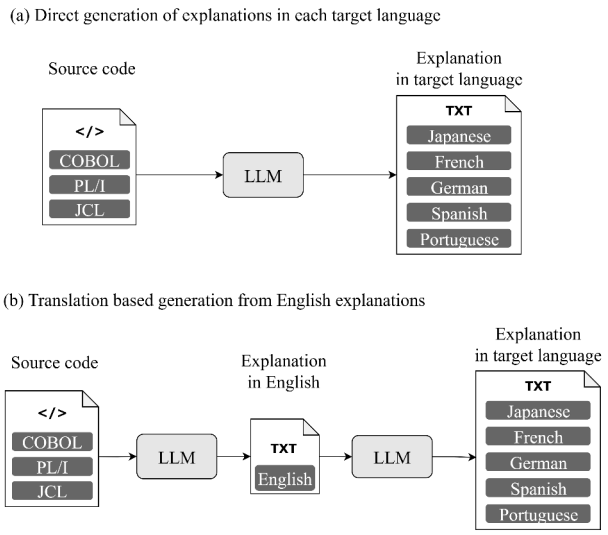

Fig. 1. Mulitilingual expalnation generation strategies.

D. Models

We evaluated two relatively lightweight language models, each with 8 billion parameters: IBM Granite 3.3 8B[1] and Meta LLamA 3.1 8B[2]. These models were selected for their compact architecture, making them suitable for on-premises enterprise deployment. In addition, we also evaluated the Mistral-Small 3.1 24B[3] model, which, despite its larger size, remains efficiently deployable in enterprise environments.

E. Evaluation

To evaluate the quality of generated explanations, we employed a combination of automatic word-overlap metrics and semantic similarity measures. Specifically, we used: BLEU [8], ROUGE-L [9], and METEOR [10]. These metrics assess lexical overlap between generated and reference explanations. To complement these, we also used sentence-level semantic similarity based on Sentence-BERT [11], which captures deeper semantic alignment beyond surface-level token matching. This combination of metrics enables a comprehensive evaluation of both linguistic fidelity and semantic accuracy across languages and explanation strategies.

IV. Results

We evaluated multilingual code explanation performance across five native languages and three legacy programming languages using lightweight and mid-scale models. We compared direct generation in each target language with translation-based generation. Explanations were assessed using both word-overlap and semantic similarity metrics (Table II—IV).

[1]https://huggingface.co/ibm-granite/granite-3.3-8b-instruc
[2]https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
[3]https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503

TABLE II
Evaluation Metrics by Language and Program for
llama-3-1-8b-instruct

Language	Program	BLEU		ROUGE-L		METEOR		Sentence Similarity	
		(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
Japanese	COBOL	0.147	**0.383**	0.299	**0.450**	0.222	**0.354**	0.739	**0.760**
	JCL	0.143	**0.341**	0.305	**0.410**	0.237	**0.351**	0.680	**0.769**
	PL/I	0.112	**0.322**	0.295	**0.443**	0.203	**0.339**	0.703	**0.808**
French	COBOL	0.228	**0.311**	0.249	**0.272**	0.269	**0.330**	0.684	**0.762**
	JCL	0.260	**0.270**	0.258	**0.259**	0.308	**0.321**	**0.775**	0.737
	PL/I	0.238	**0.295**	0.255	**0.257**	0.271	**0.341**	0.780	**0.811**
German	COBOL	0.212	**0.278**	0.180	**0.224**	0.246	**0.309**	0.705	**0.775**
	JCL	0.248	**0.254**	0.187	**0.205**	0.282	**0.291**	0.688	**0.716**
	PL/I	0.177	**0.270**	0.202	**0.207**	0.214	**0.314**	0.741	**0.836**
Spanish	COBOL	0.245	**0.305**	0.252	**0.273**	0.278	**0.334**	0.684	**0.779**
	JCL	0.285	**0.297**	0.254	**0.264**	0.304	**0.324**	**0.771**	0.763
	PL/I	0.273	**0.296**	**0.267**	0.264	0.295	**0.337**	0.770	**0.834**
Portuguese	COBOL	0.243	**0.316**	0.247	**0.276**	0.283	**0.332**	0.711	**0.777**
	JCL	0.188	**0.289**	0.194	**0.267**	0.248	**0.322**	**0.745**	0.739
	PL/I	0.203	**0.284**	**0.270**	0.254	0.249	**0.332**	0.762	**0.800**

(a): Direct generation, (b): Translation-based generation. BLEU is reported in its original 0–1 scale. Bold values indicate the higher score between (a) and (b).

TABLE III
Evaluation Metrics by Language and Program for
granite-3.3-8b-instruct

Language	Program	BLEU		ROUGE-L		METEOR		Sentence Similarity	
		(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
Japanese	COBOL	0.318	**0.337**	0.397	**0.406**	0.320	**0.418**	0.797	**0.835**
	JCL	0.302	**0.368**	0.459	**0.462**	0.313	**0.365**	**0.771**	0.764
	PL/I	0.237	**0.363**	**0.463**	0.438	0.254	**0.330**	0.793	**0.805**
French	COBOL	0.299	**0.305**	0.255	**0.262**	0.364	**0.413**	**0.859**	0.851
	JCL	**0.294**	0.279	**0.274**	0.255	**0.334**	0.315	**0.806**	0.740
	PL/I	0.281	**0.329**	0.288	**0.295**	0.289	**0.325**	0.792	**0.817**
German	COBOL	**0.258**	0.250	0.202	0.202	0.352	**0.358**	**0.842**	0.839
	JCL	0.210	**0.248**	**0.219**	0.203	0.256	**0.282**	**0.766**	0.736
	PL/I	0.199	**0.278**	0.224	**0.233**	0.234	**0.288**	0.793	**0.854**
Spanish	COBOL	0.291	**0.296**	**0.261**	0.256	0.359	**0.404**	**0.849**	0.842
	JCL	0.295	**0.296**	**0.277**	0.273	0.311	**0.321**	**0.764**	0.722
	PL/I	0.294	**0.340**	0.297	**0.302**	0.293	**0.330**	0.812	**0.821**
Portuguese	COBOL	0.240	**0.294**	0.252	**0.259**	0.305	**0.403**	0.834	**0.850**
	JCL	0.238	**0.283**	0.257	**0.280**	0.281	**0.322**	**0.796**	0.738
	PL/I	0.257	**0.327**	0.280	**0.303**	0.279	**0.318**	0.771	**0.800**

(a): Direct generation, (b): Translation-based generation. BLEU is reported in its original 0–1 scale. Bold values indicate the higher score between (a) and (b).

TABLE IV
Evaluation Metrics by Language and Program for
Mistral-Small-3.1-24B-Instruct

Language	Program	BLEU		ROUGE-L		METEOR		Sentence Similarity	
		(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
Japanese	COBOL	**0.380**	0.346	**0.418**	0.385	0.376	**0.431**	0.857	0.857
	JCL	**0.416**	0.404	**0.535**	0.517	**0.416**	0.409	**0.789**	0.788
	PL/I	0.408	**0.481**	0.521	**0.530**	0.377	**0.449**	0.863	**0.867**
French	COBOL	0.312	0.312	0.261	**0.277**	0.397	**0.429**	0.828	**0.872**
	JCL	**0.349**	0.314	**0.292**	0.283	**0.375**	0.363	**0.815**	0.789
	PL/I	0.403	**0.429**	0.325	**0.348**	0.389	**0.442**	0.796	**0.840**
German	COBOL	**0.293**	0.271	0.221	**0.223**	0.370	**0.393**	0.735	**0.865**
	JCL	0.228	**0.278**	0.212	**0.220**	0.259	**0.326**	0.736	**0.754**
	PL/I	0.369	**0.414**	0.277	**0.296**	0.352	**0.419**	0.857	**0.860**
Spanish	COBOL	0.290	**0.303**	0.243	**0.275**	0.400	**0.426**	0.812	**0.864**
	JCL	**0.350**	0.331	0.274	**0.293**	**0.402**	0.367	**0.812**	0.749
	PL/I	0.428	**0.443**	0.342	**0.357**	0.393	**0.440**	0.813	**0.847**
Portuguese	COBOL	0.297	0.297	0.257	**0.271**	0.395	**0.419**	0.845	**0.869**
	JCL	**0.328**	0.314	0.262	**0.290**	**0.378**	0.359	0.745	**0.762**
	PL/I	0.412	**0.432**	0.330	**0.349**	0.395	**0.436**	0.807	**0.843**

(a): Direct generation, (b): Translation-based generation. BLEU is reported in its original 0–1 scale. Bold values indicate the higher score between (a) and (b).

Across all models and strategies, Sentence Similarity scores were consistently high, indicating that both approaches successfully captured the semantic meaning of the explanations.

However, when evaluating using BLEU, ROUGE-L, and METOR which reflect lexical and structural quality, the scores remain moderate. This result suggest that there is still room for improvement in terms of fluency, completeness, and alignment with reference outputs.

Across both lightweight models (LLaMA and Granite), translation-based generation (b) consistently outperformed direct generation (a) across most native and programming languages and evaluation metrics, despite minor variations in absolute scores. This trend was especially evident in PL/I, where (b) was consistently better regardless of native language. In native languages such as Japanese, Spanish, and Portuguese, the advantage of (b) was more pronounced, with higher BLEU and METEOR scores across multiple program types. These results suggest that lightweight models benefit from leveraging English as an intermediate representation, likely due to stronger English-centric pretraining and limited multilingual generation capacity.

While Mistral showed more balanced performance, translation-based generation (b) still generally outperformed direct generation (a), especially in PL/I, where (b) was consistently better across all languages. However, for JCL, direct generation (a) performed better across all target languages, indicating that for certain program types, direct generation may be more effective even in medium-sized models.

In addition to these quantitative results, we examined the generated explanations more closely to understand qualitative aspects of explanation performance. In particular, we observed that accurate handling of technical terms is critical for explanation quality. Since source code contains domain-relevant identifiers and keywords, preserving their meaning across languages is essential. Misinterpretation or over-translation of such terms can significantly degrade the clarity and correctness of the explanations. For example, terms like INCLUDE, COMMAREA, and WORKING-STORAGE, which have specific roles in legacy programming languages, were sometimes over-translated. These transformations can mislead readers and reduce the technical fidelity of the explanations.

V. Conclusion

This study examined the use of lightweight language models for multilingual explanation of legacy mainframe code, focusing on model capacity, generation strategy, and language consistency. Our findings address three key research questions.

RQ1: Can lightweight language models generate high-quality multilingual explanations? Lightweight models can produce semantically accurate explanations, particularly when using translation-based strategies. However, improvements in lexical and structural quality are needed to achieve more natural and contextually appropriate outputs.

RQ2: Which strategy is more effective—direct generation or translation-based generation? Translation-based generation (b) consistently outperformed direct generation (a), especially in lightweight models and for PL/I. Direct generation showed advantages in specific cases like JCL, suggesting that strategy effectiveness may vary depending on program type and model capacity.

RQ3: Are explanation quality and generation behavior consistent across different target languages? Translation-based generation showed stable performance across languages and models. JCL favored direct generation in medium-sized models, while PL/I benefited more from translation-based generation. These trends indicate that explanation strategies can generalize well, though language- and domain-specific tuning may further improve results.

Beyond quantitative metrics, our qualitative analysis revealed that accurate handling of technical terms is critical for explanation quality. Many source code tokens are domain-specific and not intended to be translated. Over-translation of such terms can degrade clarity and reduce technical fidelity. Addressing this challenge may involve terminology-aware generation techniques, such as incorporating domain-specific knowledge from external sources or generating explanations through interactive workflows that allow users to guide or validate the treatment of technical terms.

Overall, our findings demonstrate that lightweight models, when paired with translation-based strategies, offer a practical and scalable solution for multilingual explanation of legacy code, while highlighting the importance of strategy selection and terminology preservation.

References

[1] IBM. (2025) watsonx code assistant for Z – code explanation. [Online]. Available: https://www.ibm.com/docs/en/watsonx/watsonx-code-assistant-4z/2.x?topic=explain-getting-started-code-explanation

[2] W. Sun, Y. Miao, Y. Li, H. Zhang, C. Fang, Y. Liu, G. Deng, Y. Liu, and Z. Chen, "Source code summarization in the era of large language models," *arXiv preprint arXiv:2407.07959*, 2024.

[3] Q. Peng, Y. Chai, and X. Li, "HumanEval-XL: A multilingual code generation benchmark for cross-lingual natural language generalization," *arXiv preprint arXiv:2402.16694*, 2024.

[4] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, and S. C. Hoi, "Codet5+: Open code large language models for code understanding and generation," *arXiv preprint arXiv:2305.07922*, 2023.

[5] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[6] S. Haroon, A. F. Khan, A. Humayun, W. Gill, A. H. Amjad, A. R. Butt, M. T. Khan, and M. A. Gulzar, "How accurately do large language models understand code?" *arXiv preprint arXiv:2504.04372*, 2025.

[7] M. M. Hasan, M. Waseem, K.-K. Kemell, J. Raskua, J. Ala-Rantalaa, and P. Abrahamsson, "Assessing small language models for code generation: An empirical study with benchmarks," *arXiv preprint arXiv:2507.03160*, 2025.

[8] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[9] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[10] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[11] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.