Enhancing Cross-Language Code Translation via Task-Specific Embedding Alignment in Retrieval-Augmented Generation

Manish Bhattarai
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87544
`ceodspspectrum@lanl.gov`

Minh Vu
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87544
`mvu@lanl.gov`

Javier E. Santos
Earth & Environmental Science Division
Los Alamos National Laboratory
Los Alamos, NM 87544
`jesantos@lanl.gov`

Ismael Boureima
Theoretical Division
Los Alamos National Laboratory
Los Alamos, NM 87544
`iboureima@lanl.gov`

Daniel O' Malley
Earth & Environmental Science Division
Los Alamos National Laboratory
Los Alamos, NM 87544
`omalled@lanl.gov`

Abstract

We propose a method to improve Fortran-to-C++ code translation by aligning task-specific embeddings within a Retrieval-Augmented Generation (RAG) framework. Unlike traditional retrieval approaches using generic embeddings, we align the retrieval model directly with the goal of maximizing translation quality as measured by the CodeBLEU metric, ensuring that embeddings are semantically and syntactically meaningful for this task. Utilizing 25,000 Fortran code snippets from the Stack-V2 dataset and their C++ translations generated by `llama3.1-8b`, we compute pairwise CodeBLEU scores to capture fine-grained similarities. These scores serve as supervision in a contrastive learning framework to optimize the embedding model for retrieving the most beneficial Fortran-C++ pairs. Integrating these CodeBLEU-optimized embeddings into the RAG framework significantly enhances both retrieval accuracy and code generation quality. Without fine-tuning the language model, our method improves the average CodeBLEU score from 0.64 to 0.73 (a 14% improvement) on the HPC Fortran2C++ dataset and from 0.52 to 0.60 (a 15% improvement) on the Numerical Recipes dataset, demonstrating the effectiveness and practicality of our approach.

1 Introduction

Cross-language code translation is a critical task in modern software development, especially as legacy programming languages, such as Fortran, continue to be prevalent in scientific computing, while more contemporary languages like C++ are favored for their performance and versatility in production environments. The goal of automatic translation from Fortran to C++ is to preserve the functionality and structure of legacy code while benefiting from the optimizations and ecosystem of C++. However, achieving high-quality translations that adhere to the syntax and semantic norms of the target language remains a challenging problem, particularly when there is a lack of large, aligned datasets or evaluation metrics that cover both source and target languages effectively. Traditional approaches to cross-language translation, such as Retrieval-Augmented Generation (RAG) Lewis et al. [2020] typically involve two phases: first, retrieving relevant examples from a database, followed by a language model generating code conditioned on both the query and the retrieved examples. In prior efforts, the retrieval models in RAG systems have relied on general-purpose embedding models Bhattarai et al. [2024], Li et al., which are not tailored to the specific nuances of code translation. These embeddings aim to retrieve relevant pairs from the source and target languages but do not directly optimize for the quality of the generated code. As a result, while the retrieved examples may be relevant in a broad sense, they often fail to guide the language model towards producing translations that maximize fidelity to the ground truth in the target language.

This gap is particularly problematic in scenarios where explicit metrics, such as CodeBLEU—designed to assess both syntactic and semantic correctness of translated code—are only available for the target language (e.g., C++ in this case). Without aligning the retrieval mechanism to such a task-specific metric, the system may retrieve suboptimal examples, leading to poor code generation performance. The inability to leverage task-relevant quality metrics during retrieval weakens the overall system, limiting its effectiveness in high-accuracy code translation tasks. To address these limitations, we propose a novel contrastive learning framework that aligns the retrieval phase of the RAG system with the goal of maximizing the CodeBLEU Feng et al. [2020] score for the generated C++ code. We collect a dataset of 25,000 Fortran code examples from Stack V2 Lozhkov et al. [2024] and use the `llama3.1-8b` Touvron et al. [2023] model to generate corresponding C++ translations. In the absence of ground truth C++ translations, we evaluate the quality of these translations using pairwise CodeBLEU similarity scores. This metric captures both syntactic correctness and semantic fidelity, providing a robust signal for aligning the retrieval model through contrastive learning.

Our approach directly addresses the shortcomings of general-purpose embedding models by integrating task-specific metrics into the retrieval optimization process. By aligning the retrieval model with the downstream task of producing high-quality C++ code, our method ensures that the examples retrieved during inference are not just broadly similar but are semantically and syntactically aligned in a way that enhances the LLM's generative performance. The result is a significant improvement in translation quality, as measured by CodeBLEU, over previous methods that lack such alignment.

2 Related Work

Historically, code translation strategies before the advent of large language models (LLMs) relied heavily on rule-based and statistical machine translation (SMT) systems Koehn [2009]. These systems used predefined rules or statistical mappings between the source and target programming languages, such as tree-based translation approaches that mapped syntax trees between languages. While these methods provided structured and interpretable outputs, they were limited in their ability to handle the semantic complexities of different programming languages and struggled with code diversity, edge cases, and idiomatic translations. With the rise of deep learning and LLMs, fine-tuning models on large datasets became the go-to method for improving code translation. Models like Codex Chen et al. [2021] and CodeBERT Feng et al. [2020], when fine-tuned on specific language pairs, improved translation quality by leveraging vast amounts of parallel code data. However, the main limitation of LLM fine-tuning lies in the resource-intensive process. Fine-tuning requires substantial amounts of labeled data and computational resources, making it impractical for niche or legacy languages like Fortran, where parallel data may be scarce. As a next step, task-specific alignment of LLMs emerged to improve translation by better guiding the model's output. While alignment techniques help improve output fidelity, they still necessitate fine-tuning or explicit modification of the LLM itself, which can be resource-intensive and may still fall short of generalization when translating between languages with significant structural differences Mishra et al. [2024].

RAG introduced a more flexible approach by allowing LLMs to retrieve and condition their outputs on example pairs from a relevant dataset. While RAG improves translation by augmenting the model's input, the effectiveness of this strategy depends on the quality and relevance of the retrieved examples. In an example case Bhattarai et al. [2024], the retrieval step relies on general-purpose embeddings like Nomic-Embed or CodeBERT, which, although effective at retrieving semantically

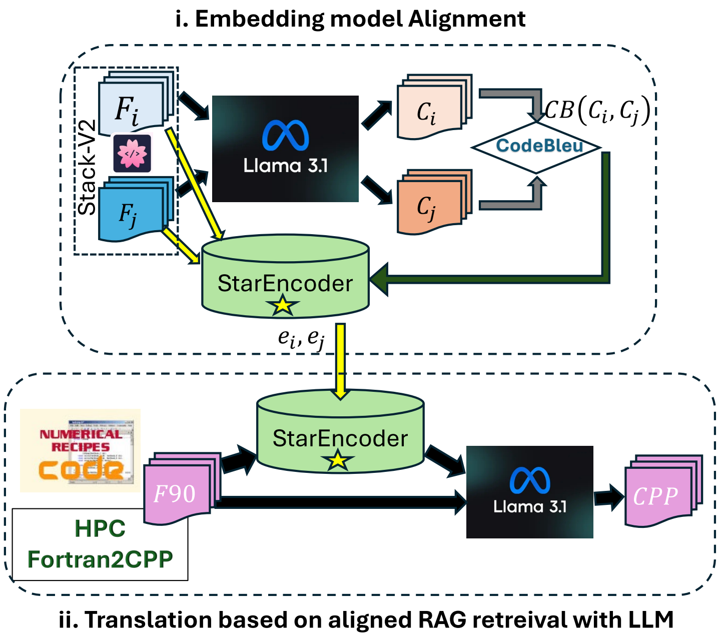

Figure 1: Overview of the proposed pipeline. i) The LLM generates pairwise code translations, which are evaluated using the CodeBLEU metric. ii) The resulting similarity scores are used to guide contrastive learning for semantic alignment of the embedding model.

similar code, are not optimized for specific downstream metrics like CodeBLEU. As a result, the LLM might not always retrieve the examples that would best assist in producing translations aligned with target-specific quality metrics.

3 Methods

Our method improves cross-language code translation from Fortran to C++ by aligning a Fortran embedding model using contrastive learning based on CodeBLEU similarity scores and integrating it within a RAG framework (Figure 1).

Code Generation and Similarity Measurement: Given a dataset of Fortran code snippets $D_f = f_1, \ldots, f_N$, we generate corresponding C++ translations $\hat{C} = \hat{c}_1, \ldots, \hat{c}_N$ using a pre-trained language model G without retrieval:

$$\hat{c}_i = G(f_i), \quad \forall i. \tag{1}$$

In the absence of ground truth translations, we compute pairwise CodeBLEU similarity scores c_{ij} between the generated translations:

$$c_{ij} = \text{CodeBLEU}(\hat{c}_i, \hat{c}_j). \tag{2}$$

CodeBLEU Ren et al. [2020] extends the traditional BLEU metric by incorporating syntactic and semantic features specific to code, providing a comprehensive similarity measure. **Embedding Function:** We define an embedding function E that maps Fortran code snippets to d-dimensional embeddings:

$$\mathbf{e}_{f_i} = E(f_i) \in \mathbb{R}^d. \tag{3}$$

Embedding Alignment with Modified InfoNCE Loss: To align these embeddings with the semantic similarities from CodeBLEU (Figure 1I), we employ a modified InfoNCE loss van den Oord et al. [2018]. We normalize the embeddings $\mathbf{h}_i = \mathbf{e}_{f_i}/|\mathbf{e}_{f_i}|$ and compute pairwise cosine similarities scaled by a temperature parameter $\tau > 0$:

$$s_{ij} = \frac{\mathbf{h}_i^\top \mathbf{h}_j}{\tau}. \tag{4}$$

The loss function is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij} \log\left(\frac{\exp(s_{ij})}{\sum_{k=1}^{N}\exp(s_{ik})}\right). \tag{5}$$

Minimizing $\mathcal{L}$ encourages embeddings of semantically similar code snippets (with higher c_{ij}) to be closer in the embedding space.

Retrieval-Augmented Generation: With the optimized embedding model E^*, we enhance code translation via a RAG framework (Figure 1II). For a query Fortran code snippet f_q, we compute its embedding $\mathbf{e}_{f_q} = E^*(f_q)$ and retrieve the top-k similar snippets f_{r_j} from D_f based on cosine similarity:

$$\text{sim}(\mathbf{e}_{f_q}, \mathbf{e}f) = \frac{\mathbf{e}_{f_q}^\top \mathbf{e}_f}{|\mathbf{e}_{f_q}||\mathbf{e}_f|}, \quad \forall f \in D_f. \tag{6}$$

Their corresponding C++ code snippets c_{r_j} augment the input to the language model G, which then generates the translated C++ code $\hat{c}_q$:

$$\hat{c}_q = G\left(f_q, (f_{r_j}, c_{r_j})_{j=1}^{k}\right). \tag{7}$$

By incorporating retrieved examples, the model produces translations that are more accurate and semantically aligned with the desired output.

4 Experiments and Results

We evaluated our method using three datasets: the HPC Fortran2CPP dataset Lei et al. [2023] with 315 Fortran-C++ code pairs, the Numerical Recipes dataset Press et al. [1988] containing 298 Fortran-C++ pairs, and the Stack-V2 dataset Lozhkov et al. [2024] comprising over 500,000 Fortran code snippets. To prepare for embedding alignment, we sampled 25,000 high-quality Fortran code snippets from Stack-V2 by selecting files larger than 500 bytes with high star and fork counts, indicating relevance and popularity. Since Stack-V2 lacks Fortran-C++ pairs, we used the `llama3.1-70b` model to extract executable Fortran code and the `llama3.1-8b` model to translate these snippets into C++ code. We computed pairwise CodeBLEU scores between the generated C++ code snippets to quantify syntactic and semantic similarities. Using these CodeBLEU metrics as supervision signals, we employed the InfoNCE loss function (Equation 5) with a temperature of 0.1 to align the embeddings from the Fortran codes, effectively training the embedding model to map semantically similar code snippets closer in the embedding space. After alignment, we integrated the optimized embedding model into the RAG pipeline, storing the Fortran-C++ pairs and corresponding embeddings in a vector database. We evaluated performance using the `llama3.1-8b` and `llama3.1-70b` models in zero-shot, 1-shot, 2-shot, and 3-shot scenarios on the HPC Fortran2C++ and Numerical Recipes datasets, following settings similar to Bhattarai et al. [2024]. CodeBLEU scores for both aligned and unaligned models were obtained by comparing the RAG-augmented generated C++ translations to the ground truth C++ code.

Table 1: Delta in Mean CodeBLEU scores between Zero- and Few-Shot prompts. The values are presented as Unaligned/Aligned scores.

		Δ in CodeBLEU scores (Unaligned / Aligned)			
Dataset	**Model**	**Zero-shot**	**1-shot**	**2-shot**	**3-shot**
HPC Fortran2++	`llama3.1 70b`	0.364	+0.262/+0.346	+0.275/+0.371	+0.281/+0.377
	`llama3.1 8b`	0.342	+0.237/+0.346	+0.261/+0.376	+0.252/+0.374
numerical_receipe	`llama3.1 70b`	0.280	+0.232/+0.313	+0.243/+0.329	+0.243/+0.317
	`llama3.1 8b`	0.276	+0.181/+0.268	+0.195/+0.292	+0.201/+0.289

Figure 2 presents scatter plots of CodeBLEU scores for code samples generated using RAG retrieval with aligned versus unaligned embeddings derived from StarEncoder. In these plots, each data point represents a single Fortran code snippet from the test set, with symbols—crosses, pluses, and triangles—indicating evaluations using 1-shot, 2-shot, and 3-shot methods, respectively. The red

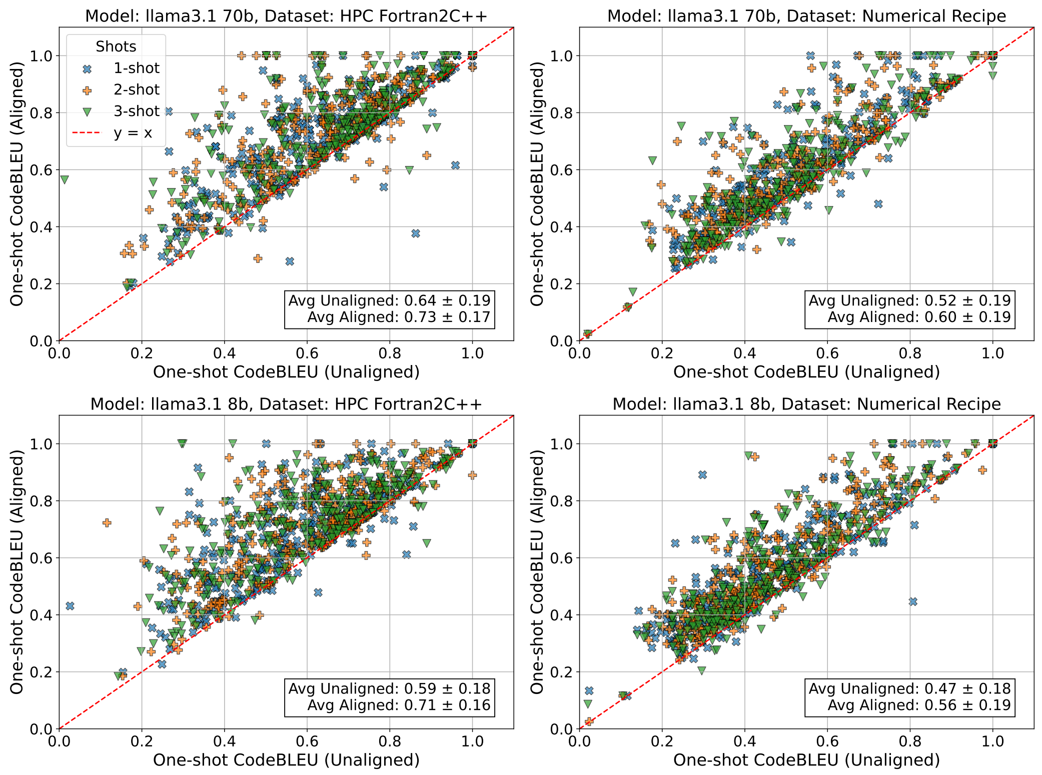

Figure 2: Scatter plots comparing the unaligned and aligned One-shot CodeBLEU scores across different shot counts (1-shot, 2-shot, 3-shot) for two models (`llama3.1 70b` and `llama3.1 8b`) and two datasets (Numerical Recipe and HPC Fortran2C++ Dataset). Each point represents a shot count, and the red dashed line represents the reference where the unaligned and aligned scores are equal. The text box in each subplot displays the average CodeBLEU performance and standard deviation for aligned vs. unaligned RAG translation across the few-shot configurations.

dashed line in each scatter plot denotes the boundary where the aligned and unaligned models yield identical CodeBLEU scores. Data points above the red line signify instances where the aligned embedding model produces translations with higher CodeBLEU scores compared to the unaligned model, indicating translations closer to the ground truth C++ code. Conversely, points below the red line represent cases where the unaligned model performs better. The observation that the majority of data points lie above the red line across all configurations demonstrates that the aligned embedding model consistently outperforms the unaligned model in translation quality. Specifically, on the HPC Fortran2C++ dataset, averaged over all shot counts and models, the aligned embeddings achieved an average CodeBLEU score of 0.73, whereas unaligned embeddings achieve 0.64. On the Numerical Recipes dataset, aligned embeddings yielded an average CodeBLEU score of 0.60, outperforming the unaligned case at 0.52.

Table 1 details the variations in mean CodeBLEU scores between zero-shot and few-shot prompting strategies for both unaligned and aligned embedding models. A key observation is that aligned models consistently exhibit greater improvements in CodeBLEU scores when transitioning from zero-shot to few-shot settings. For instance, on the HPC Fortran2C++ dataset with the `llama3.1-70b` model, the aligned embedding model achieves a delta increase of $+0.346$ in the 1-shot scenario, surpassing the unaligned model's increase of $+0.262$. At the 3-shot setting, the aligned model attains a delta of $+0.377$, exceeding the unaligned model's delta of $+0.281$. Similar patterns are observed with the `llama3.1-8b` model and on the Numerical Recipes dataset.

5 Conclusion

We presented a novel method to enhance Fortran-to-C++ code translation by aligning embeddings within a RAG framework using contrastive learning based on CodeBLEU similarity scores. This alignment brings code snippets that yield high-quality translations closer in the embedding space, enabling more effective retrieval of examples to guide the language model during code generation. Our approach significantly improved translation quality without fine-tuning the language model. Aligned embeddings increased the average CodeBLEU score from 0.64 to 0.73 (a 14% improvement) on the HPC Fortran2C++ dataset and from 0.52 to 0.60 (a 15% improvement) on the Numerical Recipes dataset. Larger models (`llama3.1-70b`) outperformed smaller ones (`llama3.1-8b`), and performance gains plateaued beyond two-shot prompting. By optimizing the retrieval mechanism through task-specific embedding alignment, our method enhances code translation efficiently and is adaptable to other tasks, especially when aligned datasets are limited or evaluation metrics like CodeBLEU are critical. Future work includes extending this strategy to more programming languages and integrating additional evaluation metrics to further improve translation quality.

References

Manish Bhattarai, Javier E Santos, Shawn Jones, Ayan Biswas, Boian Alexandrov, and Daniel O'Malley. Enhancing code translation in language models with few-shot learning via retrieval-augmented generation. *arXiv preprint arXiv:2407.19619*, 2024.

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

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.139. URL `https://aclanthology.org/2020.findings-emnlp.139`.

Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

Bin Lei, Caiwen Ding, Le Chen, Pei-Hung Lin, and Chunhua Liao. Creating a dataset for high-performance computing code translation using llms: A bridge between openmp fortran and c++. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Chuangji Li, Shizhuo Li, and Alan Wang. Retrieval-augmented multi-hop code generation with codellama and unlimiformer.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, et al. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*, 2024.

William H Press, William T Vetterling, Saul A Teukolsky, and Brian P Flannery. *Numerical recipes*. Cambridge University Press, London, England, 1988.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*, 2020.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.