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 introduce a novel method to enhance cross-language code translation from Fortran to C++ by integrating task-specific embedding alignment into a Retrieval-Augmented Generation (RAG) framework. Unlike conventional retrieval approaches that utilize generic embeddings agnostic to the downstream task, our strategy aligns the retrieval model directly with the objective of maximizing translation quality, as quantified by the CodeBLEU metric. This alignment ensures that the embeddings are semantically and syntactically meaningful for the specific code translation task. Our methodology involves constructing a dataset of 25,000 Fortran code snippets sourced from Stack-V2 dataset and generating their corresponding C++ translations using the `llama3.1-8b` language model. We compute pairwise CodeBLEU scores between the generated translations and ground truth examples to capture fine-grained similarities. These scores serve as supervision signals in a contrastive learning framework, where we optimize the embedding model to retrieve Fortran-C++ pairs that are most beneficial for improving the language model's translation performance. By integrating these CodeBLEU-optimized embeddings into the RAG framework, our approach significantly enhances both retrieval accuracy and code generation quality over methods employing generic embeddings. On the HPC Fortran2C++ dataset, our method elevates the average CodeBLEU score from 0.64 to 0.73, achieving a 14% relative improvement. On the Numerical Recipes dataset, we observe an increase from 0.52 to 0.60, marking a 15% relative improvement. Importantly, these gains are realized without any fine-tuning of the language model, underscoring the efficiency and practicality of our approach.

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

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.

This 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.

Our contribution is twofold: first, we demonstrate the effectiveness of contrastive learning for fine-tuning retrieval models in the context of cross-language code translation, using a task-specific metric to guide alignment. Second, we show that optimizing retrieval for downstream generation tasks can lead to state-of-the-art results, particularly in cases where aligned datasets are not readily available for both source and target languages. This work not only advances the field of code translation but also opens up new possibilities for applying similar techniques to other language pairs and domains where task-specific evaluation metrics are available for only one side of the translation.

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 CodeBERT Feng et al. [2020] and Codex Chen et al. [2021], 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 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.

The approach we propose offers a significant advantage by focusing on semantic alignment of the retrieval mechanism without the need to fine-tune the LLM itself. Through contrastive learning, we optimize the embedding model to retrieve Fortran-C++ pairs that are more likely to maximize the downstream metric (e.g., CodeBLEU) when used by the LLM for generation. This strategy ensures that the most relevant examples are retrieved for each translation task, improving the generation quality without requiring computationally expensive fine-tuning of the LLM. This retrieval alignment makes RAG more efficient and better suited for translating between languages where high-quality paired datasets may not be available. By concentrating on improving the quality of retrieved examples, our method achieves high-quality translation with minimal additional model training, leveraging existing LLM capabilities more effectively.

3 Methods

Our method involves aligning the Fortran embedding model using contrastive learning based on CodeBLEU similarity scores, followed by applying this aligned model within a Retrieval-Augmented Generation (RAG) framework for improved cross-language code translation from Fortran to C++ as shown in Figure 1.

Code Generation and Similarity Measurement: Given a dataset of Fortran code snippets $D_f = \{f_1, f_2, \ldots, f_N\}$, we generate corresponding C++ translations $\hat{C} = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_N\}$ using a pre-trained large language model G without retrieval:

$$\hat{c}_i = G(f_i), \quad \forall i \in \{1, 2, \ldots, N\}. \tag{1}$$

Since ground truth C++ translations are not available, we compute pairwise CodeBLEU similarity scores Ren et al. [2020] between all generated translations. For each pair $(\hat{c}_i, \hat{c}_j)$, we calculate the CodeBLEU score $S_{ij} \in [0, 1]$:

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

These scores capture the syntactic and semantic similarities between the generated C++ translations. The CodeBLEU metric extends the traditional BLEU score by incorporating syntactic and semantic aspects of code, consisting of four components: n-gram match, weighted n-gram match, syntactic AST match, and semantic data flow match. The overall CodeBLEU score S is computed as:

$$S = \alpha \cdot S_{\text{n-gram}} + \beta \cdot S_{\text{weighted n-gram}} + \gamma \cdot S_{\text{syntax}} + \delta \cdot S_{\text{semantic}}, \tag{3}$$

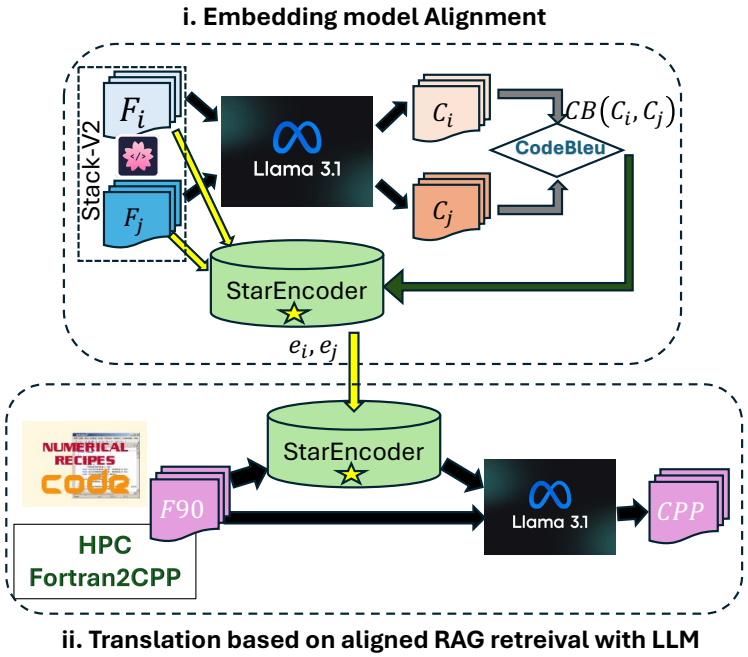

Figure 1: Overview of the proposed pipeline. i) The LLM generates pairwise code translations, which are evaluated using the CodeBLEU metric. ii) The resulting similarity scores are used to guide contrastive learning for semantic alignment of the embedding model.

where $\alpha, \beta, \gamma, \delta$ are weights summing to 1 ($\alpha + \beta + \gamma + \delta = 1$), and $S_{\text{n-gram}}, S_{\text{weighted n-gram}}, S_{\text{syntax}}, S_{\text{semantic}}$ are the scores for each component. Specifically:

- $S_{\text{n-gram}}$ is the traditional BLEU score up to n-grams.
- $S_{\text{weighted n-gram}}$ assigns weights to n-grams based on their importance.
- S_{syntax} measures the similarity between the abstract syntax trees (AST) of the code snippets.
- S_{semantic} assesses the similarity in data flow between code snippets.

Each component is calculated using algorithms tailored for code comparison, allowing for a comprehensive assessment of code similarity.

Embedding Function: Next, we define an embedding function E that maps Fortran code snippets to d-dimensional embedding vectors:

$$\mathbf{e}_{f_i} = E(f_i) \in \mathbb{R}^d, \quad \forall i \in \{1, 2, \ldots, N\}. \tag{4}$$

These embeddings are intended to capture the semantic content of the code snippets in a continuous vector space, facilitating comparison through similarity measures.

Embedding Alignment with InfoNCE Loss: To align the embedding space of code snippets with the semantic similarities measured by CodeBLEU (Figure 1I), we employ the InfoNCE (Information Noise-Contrastive Estimation) loss function van den Oord et al. [2018]. Consider a batch of N normalized embeddings $\{\mathbf{h}_i\}_{i=1}^N \subset \mathbb{R}^d$, where $\mathbf{h}_i = \mathbf{e}_{f_i}/\|\mathbf{e}_{f_i}\|$. The pairwise cosine similarities between embeddings are computed and scaled by a temperature parameter $\tau > 0$:

$$s_{ij} = \frac{\mathbf{h}_i^\top \mathbf{h}_j}{\tau}, \tag{5}$$

where s_{ij} measures the similarity between embeddings $\mathbf{h}_i$ and $\mathbf{h}_j$. The CodeBLEU similarity scores between the corresponding code snippets are denoted by $c_{ij} = S_{ij}$, forming a symmetric matrix $\mathbf{C}$.

The InfoNCE loss integrates these continuous similarity scores to weigh the contribution of each pair. The loss function is defined as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}\log\left(\frac{\exp(s_{ij})}{\sum_{k=1}^{N}\exp(s_{ik})}\right). \tag{6}$$

In Equation (6), the inner summation over k computes the normalization term for the softmax function, ensuring that the probabilities sum to one for each anchor embedding $\mathbf{h}_i$. The term inside the logarithm represents the probability p_{ij} of embedding $\mathbf{h}_j$ being similar to the anchor $\mathbf{h}_i$, given by:

$$p_{ij} = \frac{\exp(s_{ij})}{\sum_{k=1}^{N}\exp(s_{ik})}. \tag{7}$$

The InfoNCE loss in Equation (6) effectively encourages embeddings of semantically similar code snippets (those with higher c_{ij}) to have higher cosine similarities, thereby aligning them closer in the embedding space. The weighting by c_{ij} amplifies the contribution of pairs with higher semantic similarity, as determined by CodeBLEU scores.

Minimizing the InfoNCE loss $\mathcal{L}$ results in an embedding space where semantically similar code snippets are clustered together, and dissimilar ones are pushed apart. The temperature parameter τ in Equation (5) controls the concentration of the distribution; a lower τ sharpens the softmax distribution, making the model focus more on the most similar pairs.

Retrieval-Augmented Generation with Aligned Embeddings: After aligning the embedding model E^*, we integrate it into a RAG framework to enhance the code translation process (see Figure 1II). In this RAG setup, we utilize the Fortran-C++ benchmark dataset, where the Fortran code snippets are embedded using the optimized embedding model E^*.

Given a query Fortran code snippet f_q, we compute its embedding:

$$\mathbf{e}_{f_q} = E^*(f_q).$$

We then compute the cosine similarity between the query embedding $\mathbf{e}_{f_q}$ and the embeddings of all Fortran code snippets $\mathbf{e}_f$ in the dataset D_f:

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

We retrieve the top-k Fortran code snippets $\{f_{r_1}, f_{r_2}, \ldots, f_{r_k}\}$ from D_f that have the highest cosine similarity scores with $\mathbf{e}_{f_q}$ and $f_{r_i} \neq f_q$.

For each retrieved Fortran code snippet f_{r_j}, we obtain its corresponding C++ code snippet c_{r_j} from the dataset, forming Fortran-C++ pairs $\{(f_{r_j}, c_{r_j})\}_{j=1}^{k}$.

These retrieved pairs are used to augment the input to the language model G, providing additional context for code generation. The language model G then generates the translated C++ code $\hat{c}_q$ for the query f_q:

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

By incorporating the retrieved examples into its input, the language model is better equipped to produce translations that are accurate and aligned with the desired output.

4 Experiments and Results

In our study, we utilized three datasets to enhance code translation through RAG and embedding alignment. The HPC Fortran2CPP dataset Lei et al. [2023], comprising 315 Fortran-C++ code pairs, and the Numerical Recipes dataset Press et al. [1988], containing 298 Fortran-C++ pairs, were employed for RAG retrieval and evaluation with LLMs. Additionally, we used the Stack-V2 dataset Lozhkov et al. [2024], which includes over 500,000 Fortran code snippets, for RAG alignment. From Stack-V2, we sampled 25,000 high-quality and diverse Fortran code snippets by selecting files larger than 500 bytes and prioritizing those with the highest combined star and fork counts, indicating

relevance and popularity. Since Stack-V2 lacks Fortran-C++ pairs, we extracted files containing metadata, code, and comments, and utilized the `llama3.1-70b` Instruct model to extract executable Fortran code, discarding other metadata. Then, we used the `llama3.1-8b` model to translate the

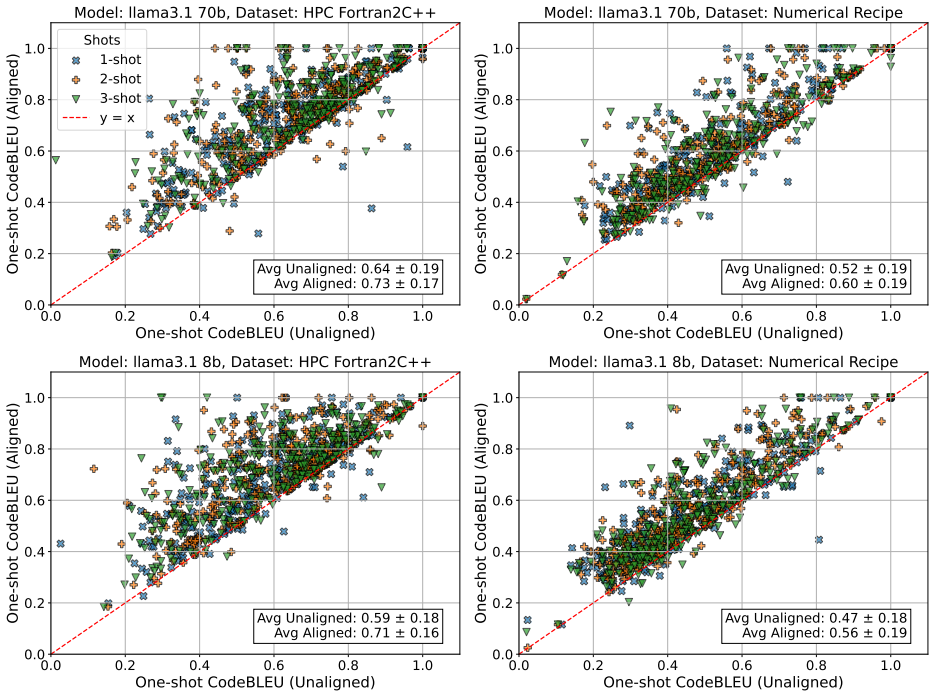

Figure 2: Scatter plots comparing the unaligned and aligned One-shot CodeBLEU scores across different shot counts (1-shot, 2-shot, 3-shot) for two models (`llama3.1 70b` and `llama3.1 8b`) and two datasets (Numerical Recipe and HPC Fortran2C++ Dataset). Each point represents a shot count, and the red 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.

cleaned Fortran code snippets into corresponding C++ code. After code translaton, we computed pairwise CodeBLEU scores between the generated C++ code snippets to quantify the syntactic and semantic similarities of their translations. Leveraging these CodeBLEU metrics and the embeddings from the Fortran codes, we employed the InfoNCE loss function van den Oord et al. [2018] with a temperature of 0.1 to align the embeddings of StarEncoder, effectively training the embedding model to map semantically similar code snippets closer in the embedding space. The embedding model was trained using the Adam optimizer with a learning rate of 1×10^{-3} and a batch size of 128 per GPU, sampling approximately 1.28 million code pairs for alignment. This training process was distributed across 256 GH200 GPUs to accelerate the process, though it can also be performed on fewer GPUs at a significantly slower pace. Post-alignment, we integrated the embedding model into the RAG pipeline, where the Fortran-C++ pairs and corresponding Fortran embeddings are stored in a vector database. We then 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 benchmark datasets HPC Fortran2C++ and Numerical Recipes, following settings similar to Bhattarai et al. [2024]. The CodeBLEU scores for both the aligned and unaligned models are obtained by comparing the RAG-augmented generated C++ translations to the ground truth C++ code.

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 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

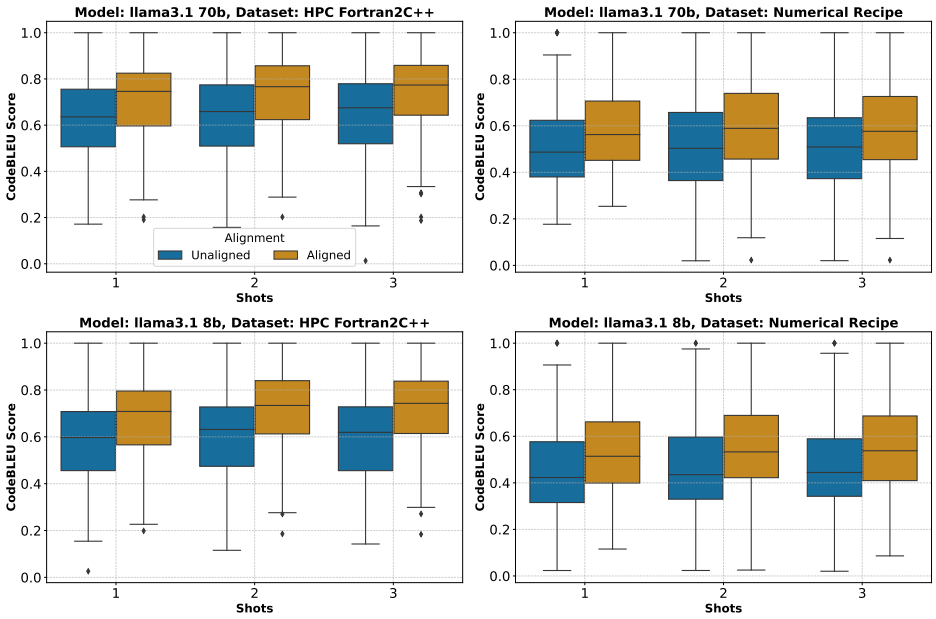

Figure 3: Box plots illustrating the distribution of CodeBLEU scores across various shot counts (1-shot, 2-shot, 3-shot) for both unaligned and aligned models. The results are presented for two models (`llama3.1 70b` and `llama3.1 8b`) across two datasets (Numerical Recipe and HPC Fortran2C++ Dataset)

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_recipes	`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

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.

In other words, the results in Figure 2 demonstrate that aligned embeddings significantly improve translation quality for each Fortran-to-C++ code translation task. 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. These substantial improvements highlight the effectiveness of our method in enhancing translation accuracy.

Figure 3 further corroborates these findings by presenting the distribution of CodeBLEU scores across various experimental configurations. The box plots reveal that aligned embeddings not only increase the median scores but also reduce performance variability. This indicates that our approach consistently enhances translation quality and leads to more reliable code translations. The consistent improvements across different model sizes (8B and 70B parameters) and datasets demonstrate the robustness and scalability of our method.

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.

These results indicate that embedding alignment significantly enhances the models' capacity to exploit few-shot prompts, leading to superior code translation performance as measured by CodeBLEU scores. Alignment optimizes the embedding space to better capture the syntactic and semantic nuances of code translation tasks, thereby augmenting the models' few-shot learning capabilities. Additionally, the larger model (`llama3.1-70b`) consistently outperforms the smaller model (`llama3.1-8b`), suggesting that increased model capacity facilitates more effective utilization of few-shot examples and alignment information. The diminishing marginal gains observed when increasing the number of shots from one to three imply that the majority of performance improvements are realized with just one or two examples, indicating diminishing returns beyond two shots.

5 Conclusion

We introduced a novel method for enhancing cross-language code translation from Fortran to C++ by aligning embeddings within a RAG framework. By leveraging contrastive learning based on CodeBLEU similarity scores, we aligned the Fortran embedding model so that code snippets yielding high-quality translations are positioned closer in the embedding space. This alignment enables the RAG system to retrieve semantically meaningful examples that effectively guide th LLM during code generation. Our experimental results demonstrate substantial improvements in translation quality without the need for fine-tuning the LLM. Specifically, using aligned embeddings increased the average CodeBLEU score from 0.64 to 0.73 on the HPC Fortran2C++ dataset and from 0.52 to 0.60 on the Numerical Recipes dataset, representing relative improvements of approximately 14% and 15%, respectively. The larger model (`llama3.1-70b`) consistently outperformed the smaller model (`llama3.1-8b`), indicating that increased model capacity enhances the effectiveness of our approach. Additionally, we observed diminishing returns beyond two-shot prompting, suggesting that most performance gains are achieved with just one or two examples. Thus, our approach significantly improves code translation performance by optimizing the retrieval mechanism through task-specific embedding alignment, rather than relying on computationally expensive fine-tuning of the LLM. This method is computationally efficient, scalable, and adaptable to other code translation tasks, particularly when aligned datasets are scarce or evaluation metrics like CodeBLEU are critical. Future work could extend this alignment strategy to additional programming languages and explore integrating other evaluation metrics to further enhance 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.