Creating a Dataset for High-Performance Computing Code Translation using LLMs: A Bridge Between OpenMP Fortran and C++

Bin Lei, Caiwen Ding
Dept. Computer Science and Engineering, University of Connecticut
Storrs, USA
{bin.lei, caiwen.ding}@uconn.edu

Le Chen, Pei-Hung Lin, Chunhua Liao
Lawrence Livermore National Laboratory
Livermore, USA
{chen142, lin32, liao6}@llnl.gov

***Abstract*—In this study, we present a novel dataset for training machine learning models translating between OpenMP Fortran and C++ code. To ensure reliability and applicability, the dataset is created from a range of representative open-source OpenMP benchmarks. It is also refined using a meticulous code similarity test. The effectiveness of our dataset is assessed using both quantitative (CodeBLEU) and qualitative (human evaluation) methods. We showcase how this dataset significantly elevates the translation competencies of large language models (LLMs). Specifically, models without prior coding knowledge experienced a boost of $\times$ 5.1 in their CodeBLEU scores, while models with some coding familiarity saw an impressive $\times$ 9.9-fold increase. The best fine-tuned model using our dataset outperforms GPT-4. It is also reaching human-level accuracy. This work underscores the immense potential of our dataset in propelling advancements in the domain of code translation for high-performance computing. The dataset is accessible at https://github.com/bin123apple/Fortran-CPP-HPC-code-translation-dataset.**

***Index Terms*—Large Language Model, Code Translation, OpenMP, Fortran, C++**

I. Introduction

The landscape of High-Performance Computing (HPC) has long been dominated by languages such as Fortran and C++, each with their unique strengths and serving different computational purposes [4]. These strengths have resulted in a rich mix of codebases spanning across these languages. However, the lack of efficient tools for translating between these two prominent languages presents a considerable challenge in the field. In response, this paper addresses the gap by introducing a novel dataset designed explicitly for the purpose of training and evaluating Large Language models (LLM) tasked with translating between OpenMP Fortran and C++.

As the world of HPC increasingly turns towards machine learning methods to optimize and enhance various computational processes, the need for effective understanding, translation, and generation of code in different languages has never been more crucial [1], [19]. Bridging the gap between Fortran and C++ by understanding their syntactic and semantic similarities and differences can contribute significantly towards this objective [7]. This understanding necessitates the creation of a reliable dataset for training and evaluating models capable of understanding and translating between these two languages - a significant stride that this paper takes.

The dataset that we introduce in this paper was created by meticulously sourcing from a diverse range of origins, including open-source projects, academic resources, and other readily available code repositories. This multifaceted approach ensures a diverse and robust dataset capable of capturing the intricacies of both languages and their translation, thus enriching its value for both model training and evaluation. By providing a comprehensive test bed, the dataset allows for the robust examination of the model's performance and offers avenues for its iterative improvement. In addition to demonstrating the practical utility of the dataset, this paper provides an in-depth analysis of its impact on models of varying complexity, from those lacking any prior coding knowledge to those already proficient in code understanding and translation. Furthermore, the paper also incorporates the results of CodeBLEU [16] and human evaluation scores, thereby providing a holistic perspective on the translation proficiency of these models post-training. Through these rigorous evaluations, the versatility of the dataset in enhancing the code translation capabilities of a broad range of models is underscored, positioning it as a valuable asset in advancing the realm of HPC code translation.

This paper seeks not only to introduce this dataset but also to lay the groundwork for future research in this promising direction. It details the creation of the dataset, its composition, and how it can be utilized for model training and evaluation. In addition, we present the results of initial experiments carried out using this dataset, demonstrating its potential to significantly contribute to the field of HPC code translation.

Envisioning this dataset to become a cornerstone for researchers and practitioners in the field, we aim to pave the way for advancements in code translation, optimization, and cross-language understanding, thereby making a substantial impact on the High-Performance Computing landscape.

Prepared by LLNL under Contract DE-AC52-07NA27344 (LLNL-PROC-851836) and supported by the Advanced Simulation and Computing Program at LLNL and the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research. This work was also funded by the University of Connecticut (Uconn).

II. Background and related work

In the realm of HPC, the conversion of legacy code to modern programming languages poses a significant challenge, often requiring substantial human intervention and expertise [17]. Legacy languages such as Fortran, which retain their foothold, particularly in the scientific computing community, often present considerable hurdles when interfaced with contemporary HPC systems [18]. The challenges stretch from issues of modern hardware utilization and integration with emerging software stacks to the absence of user-friendly interfaces and contemporary development tools.

Simultaneously, C++ has carved its niche as a leading programming language in HPC, owing to its powerful performance capabilities and object-oriented design. These features facilitate effective code organization and reuse, providing substantial advantages over legacy languages. However, the task of manually translating Fortran into C++ is a formidable, error-prone endeavor, even for seasoned programmers.

With the rise of large language models (LLMs), particularly transformer-based models such as GPT, the field of natural language processing has seen unprecedented advancements, offering successful solutions to complex tasks such as machine translation [8]. While the application of these models has primarily been in the context of human languages, their potential for extending to programming language translation is an enticing prospect.

A significant obstacle in this context, however, lies in the scarcity of high-quality, large-scale datasets that are suitable for training these predictive models on the task of code translation [14]. Existing works have largely been confined to translations between programming languages in sequential programming paradigms [9]. A clear gap exists for comprehensive datasets that focus on high-performance computing languages such as OpenMP Fortran and C++.

This paper takes a significant stride toward bridging this gap. It introduces the creation of a novel dataset specifically designed for translating OpenMP Fortran code to equivalent C++ code, with the ultimate goal of catalyzing the automation of HPC legacy code translation and modernization.

III. Dataset Creation

A. Data Collection

The sources for our dataset predominantly come from three distinct repositories: the NAS Parallel Benchmarks (NPB) [2], the Polyhedral Benchmark (PolyBench) [3], and the DataRaceBench (DRB) benchmark [11]. We have collected pairs of OpenMP Fortran vs. C++ code from these codebases, combined with manual translation as needed.

The NPB dataset, a suite designed by the NASA Advanced Supercomputing (NAS) Division, is used to evaluate the performance of parallel supercomputers [2]. The benchmarks, originally authored in Fortran, stem from computational fluid dynamics (CFD) applications and consist of five 'kernel' benchmarks along with three 'pseudo-applications.' Tasks within these benchmarks range from cubic grid assignment and successive over-relaxation to one-dimensional integration. The 'pseudo-applications' are simplified iterations of real-world computational fluid dynamics applications. Given its wide acceptance as a standard for performance comparison of parallel computers within the high-performance computing community, the NPB is an invaluable source of Fortran HPC code for our dataset. There are also C++ versions [6] derived from the official Fortran version. We paired them up at subroutine/function levels to create our new dataset.

PolyBench, short for Polyhedral Benchmark, is a compilation of programs used for extracting precise Static Control Parts (SCoPs) - elements critical to the execution of HPC and many-core architectures [3]. Extensively utilized in research revolving around polyhedral compilation and other related areas, the PolyBench suite comprises benchmarks from various computing domains, including 2D and 3D convolution, data mining, linear algebra kernels, and more. Given their compact nature, these programs are ideally suited for compiler and architecture experiments. Initially available in C, the suite now also contains versions in CUDA, OpenCL, and Fortran, making it an invaluable addition to our dataset.

DataRaceBench (DRB) is a suite of OpenMP programs specifically designed for evaluating the quality of data race detection tools [11]. It comprises a variety of OpenMP applications and kernels representing common computational patterns in scientific computing. Each benchmark within the suite is intentionally constructed either to contain or to be free of data races for testing purposes. Given its emphasis on parallelism and data interactions, the DRB serves as an excellent source of comprehensive OpenMP code patterns for our dataset. DRB also has both Fortran and equivalent C++ versions of its included OpenMP codes. This simplifies the creation of our new dataset.

B. Formatting

The process of formatting our dataset played an instrumental role in the eventual success of our model training. We aimed to ensure consistency and standardization across the Fortran and C++ code snippets, to facilitate the identification of patterns and translation rules by the predictive model.

- **Code Standardization:** All the gathered OpenMP Fortran and C++ code snippets underwent standardization using various code formatting tools. These tools automatically formatted the code to adhere to the most prevalent style guidelines in both languages. This process involved adjustments of indentations, line breaks, and modifications to variable naming and function declarations.
- **Comment Removal:** We stripped all the comments from the code snippets. Despite the integral role comments play in programming, facilitating code comprehension and functionality understanding, they can introduce noise when training a model for code translation. Thus, we decided to exclude them from our dataset.
- **Whitespace and Special Characters:** We removed all leading and trailing whitespaces from each line and replaced tabs with spaces to maintain consistency. We

also removed special characters that were not part of the syntax but user comments, such as non-ASCII characters.

- **Function Mapping:** The translation between Fortran and C++ poses a notable challenge due to the differences in function names and calling conventions between the two languages [5]. To address this issue, we created a mapping of Fortran subroutines to their corresponding C++ equivalents. This mapping was utilized to create Fotrain-C++ code pairs in our dataset. Additionally, we adopted code inlining or outlining [12] techniques as needed to match more code pairs since different implementations of the same benchmark may not have subroutines or functions at the same granularity. For example, A Fortran version of a benchmark may have a big subroutine while its corresponding C++ version has a simpler function calling another function. In this case, we can inline the callee function of the C++ version to match better with the single Fortran subroutine. Function mapping with code outlining or inlining significantly improves the quality of our Fortran-C++ code pairs.

C. Dataset Calibration

An essential aspect of our dataset creation was its calibration, which we accomplished using a similarity test to ensure the dataset's accuracy and dependability.

To gain a comprehensive understanding of the dataset's structure and consistency, we embarked on a dataset calibration process. Utilizing StarCoder [10], we generated embeddings for each Fortran-C++ pair in our dataset, followed by the computation of the cosine similarity scores for these embeddings. This procedure provided a quantitative measure of the semantic similarity between each code pair and facilitated the identification of any outliers or anomalous data points that could potentially compromise the model's learning.

This calibration process granted us a more profound comprehension of our dataset, thereby enhancing its reliability and bolstering the credibility of our subsequent analyses. We will delve into the details of the similarity test experiment in Subsection IV-B.

D. Human-level Evaluation and Test

We conducted a human-level test as a part of our dataset validation procedure to ensure its quality and practicality. This test involved expert programmers, proficient in both OpenMP Fortran and C++, conducting a manual review of a subset of the data pairs in our dataset.

These experts assessed the translations based on their correctness, readability, and how accurately they retained the semantics of the original Fortran code. They also examined potential issues that could impact the machine learning model's training, such as formatting inconsistencies, incorrect translations, or any anomalies that automated tests could not detect. The invaluable feedback garnered from this human-level testing phase significantly aided in further refining and enhancing our dataset's quality. This iterative feedback and refinement process guaranteed the creation of a high-quality, trustworthy dataset for our code translation task between OpenMP Fortran and C++.

Furthermore, we selected a random assortment of code snippets from our test set and enlisted coding professionals to translate them. We then collected and evaluated the translations provided by these experts. This approach not only served as a benchmark for assessing the performance of models fine-tuned on our dataset but also gave us a more comprehensive understanding of our dataset's complexity. We discuss the detailed results in Subsection IV-F.

IV. Experiment

With our established dataset of paired OpenMP Fortran and C++ code snippets in place, we progressed to utilize this dataset to train and/or evaluate large language models.

Our ultimate goal was to effectively execute the translation between OpenMP Fortran and C++ code. We aimed to showcase the potential of machine learning, particularly large-scale transformer models, in addressing the complex task of code translation in the high-performance computing domain.

We highlight the steps taken, the methodologies employed, and the results obtained in the subsequent subsections. This includes the model selection, model training, evaluation metrics, and experimental results along with detailed analysis.

A. Experiment setup

- **Model and Hyperparameters**: For the LLM without prior Fortran knowledge, we used models from the Open Pre-trained Transformers (OPT) [20] series for this task, which are well-known for their effectiveness in various language translation tasks. The OPT model is built on the transformer architecture, characterized by its use of self-attention mechanisms. The OPT model we've chosen utilizes a decoder-only architecture. It consists of several layers of self-attention and feed-forward neural networks. Instead of using an encoder to interpret the input code written in one language, this model directly takes the code as part of its input sequence. Utilizing the continuous representations of the input and the previously generated code, the decoder then predicts the next token for the translated code in the target programming language.
  For the LLM with prior Fortran knowledge, we used the StarCoder [10] model for analyzing a model with prior Fortran code knowledge. StarCoder, specifically designed to interpret and generate code, can process source code as input and produce an embedding that encapsulates the semantic meaning of the code. StarCoder is a model equipped with 15.5 billion parameters and features a decoder-only transformer structure. It's trained on permissively licensed code from Github, covering 80+ programming languages including both Fortran and C++. The fine-tuning was conducted on an RTX6000 GPU, utilizing PyTorch [13] version 2.01, DeepSpeed [15] 0.9.5, CUDA driver 12.1, and Cudatoolkit 11.7. The learning rate was set to 9.65e-6, the maximum sequence length was 256, and we utilized the Adam optimizer.

Beyond the models already discussed, we subjected GPT-4, the most advanced commercial large language model currently available, to evaluation on our dataset. We scrutinized the performance of GPT-4 on our test set, aiming to juxtapose its capabilities with those of other open-source models that were fine-tuned using our dataset.

- **Evaluation Metrics**: To gauge our model's performance, we employed a distinctive metric named Code Bilingual Evaluation Understudy (CodeBLEU) [16]. Specifically designed to evaluate the quality of code translation models, CodeBLEU is aptly suited for our endeavor of translating between Fortran and C++ codes. It augments the traditional BLEU (Bilingual Evaluation Understudy) score, predominantly utilized in natural language machine translation, by incorporating various code-centric features, encapsulating both syntactic and semantic elements.
  We leveraged CodeBLEU to scrutinize our model's translations, juxtaposing them with the benchmark OpenMP Fortran and C++ codes. The resultant CodeBLEU scores furnished a quantifiable metric, granting profound insights into the efficacy of our model's translations.
  CodeBLEU scores span between 0 and 1. A score of 1 epitomizes a flawless semantic match between languages. A score of 0 indicates an absolute lack of semantic coherence between the translated and the reference code.
- **Dataset Example:** Each data pair is presented in a subroutine-vs-function format. One of the data pair examples is shown in Figure1.

C++ example:
```
#include <omp.h>\n#include <stdio.h>\n\n\nint main(){\n int x = 2;\n\n #pragma omp task shared(x) mergeable\n {\n x++;\n }\n #pragma omp taskwait\n\n printf("%d\n",x);\n return 0;\n}\n
```

Fortran example:
```
program DRB130_mergeable_taskwait_orig_no\n use omp_lib\n implicit none\n\n integer :: x\n x = 2\n\n !$omp task shared(x) mergeable\n x = x+1\n !$omp end task\n\n print 100, x\n 100 format ('x ='',3i8)\nend program
```

Fig. 1. One example data pair from our Fortran-C++ code pair dataset

B. Dataset Calibration

We employed a similarity test to assess the quality of our dataset. The code similarity task is designed to measure the syntactic and/or semantic similarity between pairs of code snippets. Such an analysis is advantageous in numerous applications, including but not limited to, plagiarism detection, code reuse and refactoring, bug detection and repair, licensing compliance, and malware detection.

For each pair of code snippets within the Fortran-C++ code pair dataset, we compute a similarity score by calculating the cosine similarity of Starcoder Embedding. We utilized the similarity determined by Starcoder, an LLM trained with various programming languages (CPP and Fortran are included). Even with the out-of-box model, we observed the ability of the code to distinguish code snippets. Additionally, we manually reviewed the code during the calibration process to reassess its validity. A similarity score 1 indicates that the pair of snippets share the same functionality. Conversely, if the snippets do not share any functionality, they are assigned a score of 0. The results of this test are presented in Figure 2.

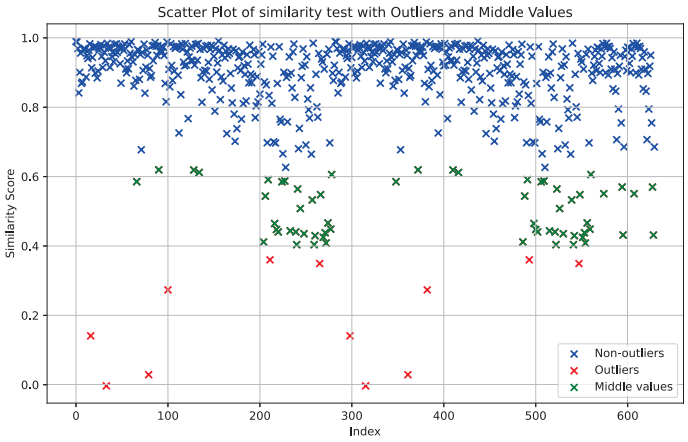

Fig. 2. Similarity test results.

We retained all data points that displayed high similarity (represented in blue) and re-evaluated and adjusted those that showcased lower similarity (indicated in green and red). Some of the red data points were discarded.

Following these adjustments, we carried out the similarity test once more to ensure that all data pairs preserved a high degree of similarity.

C. Performance on LLMs Without Prior Fortran Knowledge

In this part of the study, we evaluated large language models that had not undergone prior training on any Fortran code. This setup provided us the opportunity to examine how effectively these models can grasp and translate Fortran code, having only been trained on our dataset.

For this, we chose the OPT-1.3B [20], OPT-2.7B [20], and OPT-6.7B [20] Language Models, and trained them on our OpenMP Fortran-C++ code pair dataset. Post-training, we tested these models on a separate set of unseen C++ code snippets. The translations produced by the models were contrasted with the corresponding ground-truth Fortran code, and their performance was assessed using the CodeBLEU score.

One of the output result examples is shown in Figure 3. As can be seen, prior to fine-tuning our dataset, the OPT model entirely lacks the capacity to translate Fortran code, as illustrated in the second box from the top. However, after undergoing fine-tuning with our dataset, its capability to translate Fortran code has seen a marked improvement, as demonstrated in the third box from the top.

The CodeBLEU scores for the OpenMP C++ to Fortran code translation, performed by three variants of the OPT model before and after being fine-tuned on our dataset, are outlined in Table I.

When examining the OPT-1.3B, OPT-2.7B, and OPT-6.7B models prior to fine-tuning our dataset, the CodeBLEU scores,

C++ Source Code:
```
#include <omp.h>\n#include <stdio.h>\n#include <unistd.h>\n\nint main(int argc, char* argv[])\n{\n  int var = 0;\n  int i;\n\n  #pragma omp parallel sections\n  {\n    for (i = 0; i < 10; i++)
{\n      #pragma omp task shared(var)\n      {\n        var++;\n
}\n    }\n  }\n\n  if (var!=10) printf("%d\n",var);\n  return 0;\n}\n
```

Translated Fortran Code before Fine-tune:
```
I have a problem with the code. I have a C++ program that uses the C++ STL. I have a Fortran program that uses the C++ STL. I want to translate the C++ code to Fortran. I have a problem with the code. I have a C++ program that uses the C++ STL. I have a Fortran program that uses the C++ STL
```

Translated Fortran Code After Fine-tune:
```
program DRB073_taskshared_orig_no\n  use omp_lib\n    implicit none\n\n    integer :: var, i\n    var = 0\n\n  !$omp parallel sections\n    do i = 1, 10\n
```

Fortran GroundTruth:
```
program DRB123_taskundeferred_orig_yes\n    use omp_lib\n    implicit none\n\n    integer :: var, i\n    var = 0\n\n  !$omp parallel sections\n    do i = 1, 10\n        !$omp task
shared(var) \n        var = var+1;\n        !$omp end task\n    end do\n  !$omp end parallel sections\n\n    print 100, var\n    100 format ('var =', 3i8)\nend program
```

Fig. 3. One example answer from the original OPT-6.7B and trained OPT-6.7B model by using our Fortran-C++ code pair dataset.

derived from comparing their translated code with the ground truth, incrementally rise with the model's size, yet remain at a relatively modest level. After fine-tuning our dataset, the CodeBLEU scores observed a significant increase and continued to ascend proportionally to the model's size. Based on CodeBLEU score metrics, our dataset can enhance the Fortran code translation competency of models that lack any prior coding knowledge by a significant factor of $\times\mathbf{5.1}$ on average.

D. Performance on LLMs With Prior Fortran Knowledge

We then performed experiments involving a large language model that had already been trained on Fortran code. The aim was to understand if exposure to our dataset could enhance the translation capabilities of models that already had Fortran knowledge.

We selected a model called StarCoder [10] that had prior training on Fortran code. This model was then further trained on our Fortran-C++ code pair dataset. We evaluated the performance of this model on our test dataset. Then, we fine-tuned the model on our training dataset and used the CodeBLEU score as our primary evaluation metric.

In both experiments, we paid careful attention to the different aspects captured by the CodeBLEU score, including structural similarity, semantic and syntactic similarity, and naming consistency. This approach provided us with a comprehensive understanding of the models' translation capabilities and their ability to learn and adapt from the training on our dataset. One output result example is shown in Figure 4.

From this illustration, it is apparent that even for models already equipped with a substantial amount of coding knowledge, such as StarCoder, their proficiency in translating Fortran code remains relatively low. They are only capable of producing fragmented translations (as shown in the second box from the top). However, after fine-tuning on our dataset, their translation performance has seen a marked improvement (as demonstrated in the third box from the top).

C++ Source Code:
```
#include <omp.h>\n#include <stdio.h>\n\n#define N 100\n\nint var = 0;\n\nint main(){\n\n  #pragma omp target map(tofrom:var) device(0)\n  #pragma omp teams distribute parallel for\n for(int i=0; i<N; i++){\n    #pragma omp atomic\n    var++;\n\n #pragma omp atomic\n    var -= 2;\n  }\n\n  printf("%d\n",var);\n return 0;\n}\n
```

Translated Fortran Code before Fine-tune:
```
COMMAND ----------\n\n# MAGIC %md\n# MAGIC ## 2.2.2. OpenMP Target Offloading\n# MAGIC \n# MAGIC The OpenMP target offloading model is a new programming model that allows the programmer to explicitly offload computation to a device. The programmer specifies the device to which the computation is offloaded using the `target` clause. The `target` clause can be used with the following directives:\n# MAGIC \n# MAGIC * `target data`\n# MAGIC * `target enter data`\n# MAGIC * `target exit datan\ndisplayHTML('
```

Translated Fortran Code After Fine-tune:
```
program DRB157_atomiccritical_orig_gpu_yes\n    use omp_lib\n    implicit none\n\n    integer :: var, i\n    var = 0\n\n  !$omp target map(tofrom:var) device(0)\n  !$omp teams distribute parallel do\n    do i = 1, 100\n      !$omp atomic\n        var = var+1\n      !$omp end atomic\n\n      !$omp atomic\n        var = var-2\n      !$omp end atomic\n    end do\n  !$omp end teams distribute parallel do\n  !$omp end target\n\n    print 100, var\n    100 format ("var =",i3)\nend program
```

Fortran GroundTruth:
```
program DRB147_critical1_orig_gpu_no\n    use omp_lib\n    implicit none\n\n    integer :: var, i\n    var = 0\n\n  !$omp target map(tofrom:var) device(0)\n    !$omp teams distribute parallel do\n        do i = 0, 100\n            !$omp atomic\n  var = var+1\n            !$omp atomic\n            var = var-2\n  end do\n    !$omp end teams distribute parallel do\n    !$omp end target\n\n    print*, var\nend program
```

Fig. 4. One example answer from original StarCoder and trained StarCoder model by using our Fortran-C++ code pair dataset.

The CodeBLEU scores for StarCoder are shown in Table I. We can observe that for models that possess prior coding knowledge, after fine-tuning on our dataset, the CodeBLEU scores obtained by comparing their translated code with the Groundtruth show a significant increase. Moreover, after fine-tuning, their Fortran code translation proficiency surpasses that of models without any prior coding knowledge, even though the Fortran translation proficiency of the two types of models was roughly equivalent before fine-tuning. In conclusion, based on the CodeBLEU score metrics, our dataset can significantly amplify the Fortran code translation proficiency of models with prior coding knowledge by a factor of $\times\mathbf{9.9}$. Alongside this, we introduced an evaluation using the advanced language model, GPT-4, which yielded a comparable score of 0.56. The StarCoder model was able to exceed the performance level of GPT-4 once it was fine-tuned using our meticulously curated dataset. Thereby demonstrating the value and effectiveness of our dataset.

TABLE I
CODEBLEU SCORE ANALYSIS OF LANGUAGE MODELS FOR FORTRAN-C++ TRANSLATION

	OPT-1.3B	OPT-2.7B	OPT-6.7B	StarCoder	GPT-4
Original	0.0328	0.0513	0.0720	0.0619	0.560
Trained	0.221	0.248	0.254	**0.613**	N/A
Ratio	× 6.73	× 4.83	× 3.53	× **9.90**	N/A

E. Evaluation of Models by Human

In addition to the evaluation using the CodeBLEU metric, we also performed a human evaluation to assess the quality of the translations produced by our model. A panel of expert

programmers proficient in both OpenMP Fortran and C++ were recruited to review a random sample of the translated code snippets.

Each reviewer was tasked with assessing the translations, considering the correctness, readability, and functionality. These scores were then averaged to produce a final rating for each translated code snippet. Each expert independently evaluated 25% of the generated code, scoring it from 0 to 5 shown in Table II. The assessment results were essentially consistent with those obtained using the CodeBLEU score. After training on our dataset, the model's ability to translate between OpenMP Fortran and C++ code significantly improved. Notably, for models that already have a certain level of code knowledge, such as StarCoder, the improvement in their code translation capabilities was even more prominent.

In the human evaluation, the fine-tuned StarCoder model achieved scores that were almost on par with those of GPT-4, further illustrating the substantial efficacy of our dataset-specific fine-tuning.

TABLE II
ASSESSMENT OF LANGUAGE MODELS FOR FORTRAN-C++ TRANSLATION: **EXPERT EVALUATION SCORES** RANGING FROM 0 TO 5.

	OPT-1.3B	OPT-2.7B	OPT-6.7B	StarCoder	GPT-4
Original	0.17	0.10	0.27	0.30	4.72
Trained	2.13	2.23	2.23	**4.77**	N/A
Ratio	× 12.5	× 22.3	× 8.26	× **15.9**	N/A

F. Evaluation of Humans as Players

Due to the diverse methods of code translation, the final translation result might not necessarily match the ground truth 100%. Thus, it was necessary to establish a human-level benchmark CodeBLEU score. This benchmark serves to represent the level at which a model's CodeBLEU score, derived from its code translation, aligns with that of a manual translation for this specific task. Therefore, we engaged a panel of coding experts to select and translate a diverse range of codes from our test set. The translated codes were then compared against the Groundtruth values using the CodeBLEU score metric. This process enabled us to assess a benchmark to gauge the performance of the fine-tuned models. One example of this experiment is shown in Figure 5.

In this illustration, we have included the following components: the original C++ code and the code translation results of manual code translation. In this experiment, we compared the results of manual code translation with the ground truth to obtain a benchmark score using CodeBLEU. The final results are shown in Table III.

TABLE III
CODEBLEU SCORES EVALUATION OF CODE GENERATED BY HUMAN EXPERTS

	Score
Evaluation #1	0.556
Evaluation #2	**0.657**
Evaluation #3	0.558

C++ Source Code:
```
#include <stdio.h>\nvoid foo()\n{\n  static int q; \n  q += 1;\n}\n\nint main()\n{ \n  #pragma omp parallel \n  {\n  foo();\n  }\n  return 0;   \n}\n\n
```

Translated Fortran Code before Fine-tune:
```
')\n
self.assertEqual(self.translator.translate(), '#include <omp.h>\nuse
omp_lib\n\nprogram foo\n  integer :: q\n  q = 0\n !$omp parallel\n !$omp
do\n  do q = 1, 1000000000\n  end do\n !$omp end do\n !$omp end
parallel\nend program foo\n')\n\n    def test_omp_for_loop(self):\n
self.translator.set_source_text('#pragma omp parallel for\nfor (int i =
0; i < 1000000000; i++)\n{\n  printf("Hello World");\n}\n\n')\n
self.assertEqual(self.translator.translate(), '#include <omp.h>\nuse
omp_lib\n\nprogram foo\n  integer :: i\n !$omp parallel do\n  do i = 1,
1000000000\n    print *, "Hello World"\n  end do\n !$omp end parallel
do\nend program foo\n')\n\n    def
test_omp_for_loop_with_private(self):\n
self.translator.set_source_text('#pragma omp parallel for
private(i)\nfor (int i = 0; i < 100000000
```

Translated Fortran Code After Fine-tune:
```
program DRB077_static_orig_yes\n
use omp_lib\n    implicit none\n\n   !$omp parallel\n    call foo()\n
!$omp end parallel\ncontains\n    subroutine foo()\n        static
integer :: q\n        q = q+1\n    end subroutine foo\nend program
```

Fortran GroundTruth:
```
program DRB082_declared_in_func_orig_yes\n    use
omp_lib\n    use global_foo\n    implicit none\n\n    !$omp parallel\n
call foo()\n    !$omp end parallel\nend program
```

Human Answer:
```
subroutine foo\ninteger, save :: q\nq = q+ 1\nend
subroutine\n\nPROGRAM main\nUSE OMP_LIB\n!$omp parallel\ncall foo\n!$omp
end parallel\n\nend program
```

Fig. 5. One example answer of the human evaluation experiment.

Following a comprehensive series of three manual evaluations conducted by seasoned experts in both Fortran and C++ languages, the average CodeBLEU score achieved was recorded at 0.657. More impressively, the CodeBLEU performance (0.613 as shown in Table I) of the StarCoder model, post-fine-tuning, was found to be on par with the evaluation results produced by our human experts.

This promising outcome provides substantial evidence underscoring the effectiveness of our uniquely designed dataset. It clearly illustrates the capability of our dataset to serve as a powerful tool in enhancing the translation proficiency between Fortran and C++ within the realm of High-Performance Computing (HPC).

V. CONCLUSION

To conclude, we have developed a unique dataset tailored for translating between OpenMP Fortran and C++ in the high-performance computing domain. This dataset significantly amplifies the translation capacities of language models, exhibiting an enhancement factor of $\times\mathbf{5.1}$ in their CodeBLEU scores on average for models without prior coding knowledge and $\times\mathbf{9.9}$ for models with some coding familiarity. The best fine-tuned model using our dataset outperforms GPT-4. It is also reaching human-level accuracy.

These marked improvements underline the power of our dataset to advance the field of Fortran and C++ HPC code translation. Notably, our work represents a valuable asset for ongoing research in this area, providing a rigorous foundation for models learning code translation. Hence, it sets a promising stage for future breakthroughs in this realm and highlights the importance of our contribution to the community.

REFERENCES

[1] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74, 2021.

[2] David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The NAS parallel benchmarks—summary and preliminary results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, pages 158–165, 1991.

[3] Prasanth Chatarasi, Jun Shirako, Martin Kong, and Vivek Sarkar. An extended polyhedral model for SPMD programs and its use in static data race detection. In *Languages and Compilers for Parallel Computing: 29th International Workshop, LCPC 2016, Rochester, NY, USA, September 28-30, 2016, Revised Papers 29*, pages 106–120. Springer, 2017.

[4] Paweł Czarnul, Jerzy Proficz, and Krzysztof Drypczewski. Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems. *Scientific Programming*, 2020:1–19, 2020.

[5] J Ewer, B Knight, and D Cowell. Case study: an incremental approach to re-engineering a legacy fortran computational fluid dynamics code in C++. *Advances in Engineering Software*, 22(3):153–168, 1995.

[6] Dalvan Griebler. NPB-CPP: A C++ version of the NAS Parallel Benchmarks, 2023. GitHub repository.

[7] Ralf W Grosse-Kunstleve, Thomas C Terwilliger, Nicholas K Sauter, and Paul D Adams. Automatic Fortran to C++ conversion with FABLE. *Source code for biology and medicine*, 7(1):1–11, 2012.

[8] Amr Hendy, Mohamed Abdelrehim, Amr Sharaf, Vikas Raunak, Mohamed Gabr, Hitokazu Matsushita, Young Jin Kim, Mohamed Afify, and Hany Hassan Awadalla. How good are gpt models at machine translation? a comprehensive evaluation. *arXiv preprint arXiv:2302.09210*, 2023.

[9] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. *arXiv preprint arXiv:2006.03511*, 2020.

[10] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

[11] Chunhua Liao, Pei-Hung Lin, Joshua Asplund, Markus Schordan, and Ian Karlin. DataRaceBench: a benchmark suite for systematic evaluation of data race detection tools. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2017.

[12] Chunhua Liao, Daniel J Quinlan, Richard Vuduc, and Thomas Panas. Effective source-to-source outlining to support whole program empirical optimization. In *Languages and Compilers for Parallel Computing: 22nd International Workshop, LCPC 2009, Newark, DE, USA, October 8-10, 2009, Revised Selected Papers 22*, pages 308–322. Springer, 2010.

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[14] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

[15] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506, 2020.

[16] 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.

[17] Jeffrey P Slotnick, Abdollah Khodadoust, Juan Alonso, David Darmofal, William Gropp, Elizabeth Lurie, and Dimitri J Mavriplis. CFD vision 2030 study: a path to revolutionary computational aerosciences. Technical report, 2014.

[18] Thomas Sterling, Maciej Brodowicz, and Matthew Anderson. *High performance computing: modern systems and practices*. Morgan Kaufmann, 2017.

[19] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. AI for Science: Report on the Department of Energy (DOE) Town Halls on Artificial Intelligence (AI) for Science. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[20] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.