The Journal of Systems and Software 212 (2024) 112002

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Few-shot code translation via task-adapted prompt learning[☆]

Xuan Li, Shuai Yuan, Xiaodong Gu, Yuting Chen, Beijun Shen[*]

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China

ARTICLE INFO

Dataset link: https://github.com/yssjtu/code-translation

Keywords:
Code translation
Parameter-efficient fine-tuning
Few-shot learning
Task-adapted prompt tuning
Pre-trained language model

ABSTRACT

Pre-trained models such as CodeT5 and TransCoder have achieved impressive progress in software engineering. However, fine-tuning PLMs for code translation is confronted with significant challenges owing to the scarce availability of parallel code. Large language models such as ChatGPT have exhibited considerable promise in few-shot learning where only a small number of demonstration examples are given to the LLM. Yet they have not been specifically optimized for domain-specific tasks, and their use often entails significant manual effort in manually curating prompts. In this paper, we propose FSCTrans, a novel parameter-efficient tuning approach for code translation when furnished with only a few demonstration examples. (1) to efficiently reuse prior knowledge during pre-training, FSCTrans employs task-adapted prompt tuning, which freezes the pre-trained CodeT5 while merely updating parameters in a small prompt module; (2) to enable parameter efficient tuning on only a small number of examples, FSCTrans bridges pre-training to the translation task through a new pre-training objective of code-to-code generation. We evaluate FSCTrans on Java↔Python and Java↔C# datasets from both real-world projects and online judge problems. The evaluation results show that FSCTrans is remarkably effective in few-shot code translation: on average, it improves CodeT5 by 54.61% and 31.59% in terms of BLEU-4 and CodeBLEU; notably, FSCTrans demonstrates 14.42% and 18.36% superior performance in Java → C# translations in terms of BLEU-4 and CodeBLEU compared to ChatGPT.

1. Introduction

Code translation, the task of porting source code written in one language to another, has become an emerging and rapidly growing technology in software engineering (Jiao et al., 2023b). Software is often developed for multiple platforms, which require the same function to be written in different programming languages. Migrating a codebase to a new language can greatly facilitate multilingual development. However, manually porting code to another language is tedious and expensive. For example, migrating the COBOL-based platform of the Commonwealth Bank of Australia to Java lasted for five years and cost around 750 million dollars (Rozière et al., 2020). As such, automated code translation is strongly demanded.

In recent years, code translation has been advanced significantly by pre-trained language models (PLMs) such as CodeBERT (Feng et al., 2020), CodeT5 (Wang et al., 2021), and TransCoder (Rozière et al., 2020). By pre-training on a large-scale monolingual code corpus, PLMs learn the generic knowledge of multiple programming languages. This enables a more efficient adaptation to the downstream code translation task through fine-tuning on a small parallel code corpus.

However, fine-tuning PLMs to code translation is notably challenging due to the lack of parallel code pairs. Unlike natural languages, which have tens of millions of parallel resources (e.g., the OPUS-100 (Zhang et al., 2020) contains 55M parallel texts in 99 language pairs), the availability of bilingual code is far more limited: writing massive parallel code is prohibitively expensive and impractical. For this reason, there have been a small number of projects on GitHub that have multilingual releases. For example, the state-of-the-art benchmark for code translation in CodeXGLUE (Lu et al., 2021) only contains 10,000 Java-C# parallel code that was extracted from four bilingual projects. This is even more challenging in domain-specific or newly-created languages where parallel resources can hardly be obtained. These languages lack comprehensive parallel corpora, and constructing a sufficient corpus for fine-tuning would require a substantial amount of manual effort.

One promising solution is few-shot learning with large language models (LLM) such as ChatGPT (OpenAI, 2023; Brown et al., 2020), where we prepend a few demonstrations (i.e., input–output examples) into the model's context window and let the model generate the desired output. However, direct few-shot learning on LLMs usually requires

☆ Editor: Dr Burak Turhan.
* Corresponding author.
E-mail addresses: riken01@sjtu.edu.cn (X. Li), yssjtu@sjtu.edu.cn (S. Yuan), xiaodong.gu@sjtu.edu.cn (X. Gu), chenyt@sjtu.edu.cn (Y. Chen), bjshen@sjtu.edu.cn (B. Shen).

https://doi.org/10.1016/j.jss.2024.112002
Received 13 September 2023; Received in revised form 3 January 2024; Accepted 10 February 2024
Available online 17 February 2024

a set of carefully curated prompts by humans, which is costly and inefficient. Recent advances in in-context learning address this problem by automatically searching demonstrative examples from a candidate pool (Hu et al., 2022a). However, building such an example pool is still costly in the code translation task. Furthermore, although LLMs deliver outstanding performance across a wide range of tasks, they are typically designed for open-domain tasks, which can result in suboptimal performance in specific domains, particularly with low-resource corpora (Jiao et al., 2023a).

In this paper, we present FSCTrans, a novel parameter-efficient tuning approach for code translation when furnished with only a few demonstration examples. FSCTrans adapts the pre-trained CodeT5 model to the translation task through prompt tuning. It prepends prefix tokens to the encoder and decoder of CodeT5 and merely updates the parameters in the prompt module while freezing the entire CodeT5 model to retain the pre-trained knowledge. To enable prompt tuning on a small number of demonstration examples, we further introduce a new pre-training objective called code-to-code generation. The objective partitions a code snippet into two subsequences and prompts CodeT5 to generate the subsequent code based on the provided partial code. This adaptation aligns the monolingual pre-training objectives of CodeT5 with the parallel code translation task.

To evaluate FSCTrans, we build a few-shot code translation dataset that involves examples of four translations (i.e., Java→Python, Python→Java, Java→C#, and C#→Java) from real-world projects (Lu et al., 2021) and online judge problems (GeeksforGeeks, 2023). We then compare the performance of FSCTrans against typical methods, including simple copy, Transformer, CodeBERT, CodeT5, TransCoder, SoftPrompt, and LoRA. Experimental results show that FSCTrans remarkably outperforms the state-of-the-art code translation models when only a small number of examples are given. On average, FSCTrans improves the supervised model CodeT5 by 54.61% and 31.59% in terms of BLEU-4 and CodeBLEU. FSCTrans also exhibits better performance than unsupervised methods such as Transcoder, and traditional parameter-efficient tuning methods such as LoRA in the few-shot setting.

Our contributions can be summarized as follows:

- To the best of our knowledge, we are the first to propose few-shot code translation via few-shot learning, which tackles the low-resource problem of code translation in the supervised setting.
- We propose a novel *task-adapted prompt learning* framework to mitigate the gap between pre-training and few-shot code translation. Our task adaptation technique enables prompt learning to be effective in the code translation task.
- We evaluate FSCTrans on a few-shot parallel codebase collected from real-world projects and online judge problems. The results show that FSCTrans outperforms the state-of-the-art methods by a remarkable margin.

2. Background

2.1. Code translation based on PLMs

Recently, code translation has been advanced by pre-trained models significantly. CodeBERT (Feng et al., 2020) is the first attempt to apply PLM for programming languages. CodeBERT is built upon BERT, an encoder-only model with Transformer. Two pre-training objectives are applied to the model, namely, masked language modeling (MLM) and replaced token detection (RTD). CodeBERT can be integrated into a sequence-to-sequence model for code translation (Lu et al., 2021). For example, we can initialize the Transformer encoder using the pre-trained CodeBERT and then train a randomly initialized decoder from scratch, since CodeBERT is an encoder-only model.

Another effective code PLM is CodeT5 (Wang et al., 2021), which employs an encoder–decoder architecture. CodeT5 is pre-trained using four tasks: (1) masked span prediction (MSP), (2) identifier tagging (IT), (3) masked identifier prediction (MIP), and (4) bimodal dual generation (BDG). CodeT5 can be naturally adapted for code translation owing to its sequence-to-sequence architecture. Compared to CodeBERT, CodeT5 does not need to train an additional Transformer decoder from scratch. Thus, the pre-trained CodeT5 can be directly fine-tuned on a parallel code corpus to achieve the state-of-the-art performance (Wang et al., 2021).

With the advent of large-scale models, GPT-based models such as ChatGPT (OpenAI, 2023) have also been applied to the field of code translation and have demonstrated exceptional performance (Jiao et al., 2023b).

2.2. Few-shot learning and prompt learning

Few-shot learning is a deep learning technique that aims at transferring a trained model to a new task with a small number (e.g., <100) of examples (Wang et al., 2020). Deep learning models are usually data hungry and require to be trained on a large amount of data. This restricts their applicability on scarce domain-specific data (Fu and Menzies, 2017). Few-shot learning alleviates this problem by casting a pre-trained model to a new task using a small number of examples. This significantly reduces the cost of constructing a large-scale dataset.

Prompt learning has been an emerging trend for few-shot learning with PLMs (Lester et al., 2021; Cui et al., 2022). Prompt learning aims to find a proper context that steers the PLM to adapt its knowledge to a specific downstream task with only a small number of examples. It was initially introduced in the popular GPT-3 model (Brown et al., 2020). Instead of fine-tuning a pre-trained model on downstream data, GPT-3 generates the desired outputs by directly prepending a natural language prompt (e.g., "Translate from English to French:" for the translation task) and a few examples to the PLM input (Brown et al., 2020).

Prompt learning casts downstream tasks to the same form as the pre-training objectives by appending prompts into the PLM input. This enables the PLM to generate the target result with minor adjustments, thus encouraging downstream tasks to reuse knowledge from PLMs. As manually designed prompts can be error-prone, researchers have also proposed continuous prompt learning, which automatically learns to construct soft prompt templates (Shin et al., 2020; Liu et al., 2023b; Hambardzumyan et al., 2021). For example, Gao et al. (2021) proposed a prompt-based framework that searches for the optimal task-specific template automatically and uses it to transform a classification task into the MLM task. Rather than merely tuning the embeddings of prompt tokens, Li and Liang (2021) proposed prefix-tuning, where a prefix module is inserted to affect the activation layers of the PLMs directly. It prepends a sequence of continuous task-specific vectors (prefix) to the input. These prefix vectors are initialized as a trainable matrix that can be propagated upward to all Transformer activation layers in PLMs and rightward to subsequent tokens. In contrast to fine-tuning which updates all PLM parameters, prefix-tuning merely optimizes parameters in the prefix module upon the same training objective. The trained prefix module can be applied to all instances of the task.

3. Approach

In order to achieve low-resource code translation, we propose FSCTrans, a novel code translation approach with task-adapted prompt learning. The main goal of FSCTrans is to adapt a code PLM to the downstream task so that the translation model can reuse knowledge in the pre-trained model while only requiring a small number of examples.

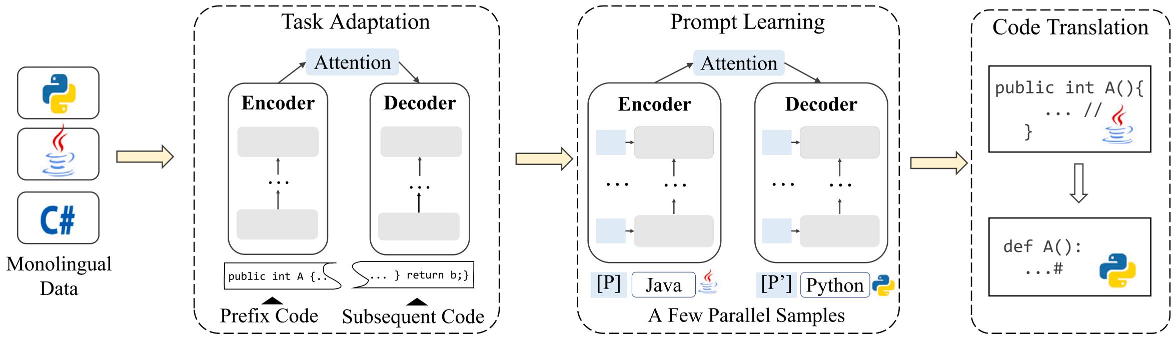

Fig. 1. An overview of FSCTrans.

3.1. Overview

Fig. 1 illustrates the framework of FSCTrans. Overall, FSCTrans takes CodeT5 (Wang et al., 2021) as the backbone PLM, and extends it with task-adapted prompt learning. CodeT5 is a sequence-to-sequence model, which conforms to the code-to-code generation scenario in task adaption. More importantly, it achieves the best code translation results across a range of baselines in our experiments. The pipeline involves three steps.

- First, we adapt the pre-training objectives of CodeT5 to the translation task through a pre-training objective of code-to-code generation to enable prompt learning, which generates the follow-up sequence given a prefix code snippet.
- Second, we transfer the pre-training CodeT5 to few-shot code translation through prompt learning, which prepends prefix tokens to the encoder and decoder of CodeT5. Prompt learning merely optimizes the prompt parameters while freezing the pre-trained model.
- Finally, we perform code translation on the FSCTrans model. The details of the three steps are elaborated in the following subsections respectively.

3.2. Prompt learning

In some specific domains or languages, parallel code is very limited. Directly fine-tuning a PLM on parallel code is ineffective when only a small number (e.g., <100) of examples are given because the pre-trained model tends to overfit the translation data (Li and Liang, 2021). In addition, fine-tuning all parameters of the model may cause catastrophic forgetting of the model (Kirkpatrick et al., 2017).

To efficiently reuse PLMs for few-shot code translation, we employ prompt tuning (Li and Liang, 2021), which is a lightweight alternative to fine-tuning. As opposed to fine-tuning which retrains the PLM in the downstream task, prompt learning aims at finding a proper context of the input that steers the PLM to adapt its knowledge to a specific downstream task. For this purpose, it freezes the parameters of the PLM while merely learning to adjust a sequence of prompt tokens accompanied by inputs. By freezing the PLM, the pre-trained knowledge is preserved, and the number of trainable parameters is significantly reduced. This enables the model to generalize better to unseen test data.

Fig. 2 illustrates the architecture of our prompt-based model. Let $x = (x_1, \dots, x_N)$ be a code snippet in the source language, and $y = (y_1, \dots, y_T)$ be the translated code in the target language. The prefix module prepends a sequence of K prompt tokens $P = (P_1, \dots, P_K)$ and $P' = (P'_1, \dots, P'_K)$ for both the source and the target sequences, obtaining $\tilde{x} = [P; x]$ and $\tilde{y} = [P'; y]$. The resulting pairs ($\tilde{x}$ and $\tilde{y}$) are fed into the encoder and decoder of CodeT5, respectively. Here, the inserted prompt tokens act as a mutual context that steers the PLM to be adapted to a specific downstream task without altering its pre-trained parameters.

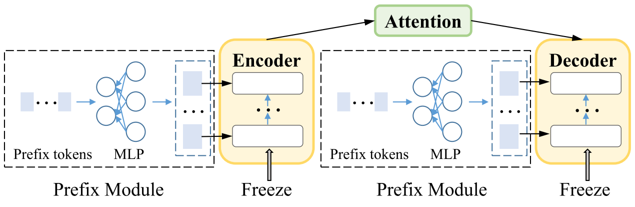

Fig. 2. Model architecture of prompt learning.

Same as code tokens, prompt tokens are initially embedded into vectors:

$$\begin{aligned}
\mathbf{e}(P_k) &= \mathbf{W} P_k, \quad k = 1, \dots, K \\
\mathbf{e}(x_i) &= \mathbf{W} x_i, \quad i = 1, \dots, N \\
\mathbf{e}(P'_k) &= \mathbf{W}' P'_k, \quad k = 1, \dots, K \\
\mathbf{e}(y_t) &= \mathbf{W}' y_t, \quad t = 1, \dots, T
\end{aligned} \tag{1}$$

where $\mathbf{W}$ and $\mathbf{W}'$ denote the trainable embedding matrices. Each P_k or P'_k = one-hot (k) is a K-dimensional vector where only the kth dimension is 1 and the other dimensions are zero.

The prefix embeddings are transformed into the prefix (past) hidden states for both the encoder and decoder using two fully connected neural networks, namely, f_θ and g_ϕ, respectively:

$$\begin{aligned}
\mathbf{H}^p_{1:K} &= f_\theta(\mathbf{e}(P_1)), \dots, f_\theta(\mathbf{e}(P_K)) \\
\mathbf{S}^p_{1:K} &= g_\phi(\mathbf{e}(P'_1)), \dots, g_\phi(\mathbf{e}(P'_K))
\end{aligned} \tag{2}$$

where $\mathbf{H}^p$ and $\mathbf{S}^p$ stand for the prefix (past) hidden states for the Transformer encoder and decoder respectively; θ and ϕ represent the trainable parameters for the two prompt modules.

The two sequences of pseudo hidden state $\mathbf{H}^p$ and $\mathbf{S}^p$ have the same shape as Transformer's. They are passed into corresponding Transformer layers of CodeT5 as the initial hidden states to generate the follow-up hidden states.

$$\begin{aligned}
\mathbf{H}_i &= \mathrm{Enc}([\mathbf{H}^p_{1:K}; \mathbf{H}_{1:i-1}], \mathbf{e}(x_i)), \quad i = 1, \dots, N \\
\mathbf{S}_t &= \mathrm{Dec}([\mathbf{S}^p_{1:K}; \mathbf{S}_{1:t-1}], \mathbf{H}_{1:N}, \mathbf{e}(y_t)), \quad t = 1, \dots, T
\end{aligned} \tag{3}$$

where $\mathbf{S}^p_{1:K}$ and $\mathbf{S}_{1:t}$ denote the hidden states of the Transformer decoder.

Finally, the decoder generates the target code tokens sequentially based on the hidden states:

$$p(y_t | y_{<t}, x) = \mathrm{softmax}(\mathbf{W}^o \mathbf{S}_t), \quad t = 1, \dots, T \tag{4}$$

where $\mathbf{W}^o$ stands for the output layer in the CodeT5 decoder. By inserting the prompt encodings before the activation layer, the prompt tokens have the potential to adjust the PLM more effectively.

The training objective for code translation is the same as that of the original CodeT5, i.e., minimizing the cross-entropy loss:

$$L(\theta, \phi | x, y, P_{1:K}, P'_{1:K}) = -\sum_{l=1}^{|D|} \sum_{t=1}^{T} \log(p(y_t | x)) \tag{5}$$

where $|D|$ represents the size of the dataset; θ and ϕ are the trainable parameters.

It is worth noting that while prompt learning has a similar objective function to CodeT5, the number of trainable parameters (i.e., θ and ϕ) is only 8.9% of that of the original CodeT5 model.

3.3. Task adaptation from pre-training to translation

Despite mitigating overfitting, directly applying prompt learning to code translation is challenging when it comes to the few-shot setting. One of the obstacles is the heterogeneity between the translation task and the pre-training objectives of the PLM. Code translation is a code-to-code generation task, while PLMs such as CodeT5 are pre-trained with masked language modeling and NL-PL dual generation. They have never seen any code-to-code pair during pre-training. Prior research (Gu et al., 2022; Chada and Natarajan, 2021) has shown that the more similar the downstream task is to the pre-training objectives, the more likely it can leverage pre-trained knowledge. On the other hand, a small number of code examples are insufficient to mitigate the heterogeneity between pre-training and fine-tuning. Hence, we need explicit adaptations to exploit the monolingual data more efficiently.

In light of this, we extend prompt learning with a task adaptation step. To align pre-training with translation, we construct a pre-training objective of code-to-code generation on large monolingual code corpora only, namely, predicting the subsequent code given a partial code snippet. Let $x = (x_1, \dots, x_N)$ be a code snippet that consists of a sequence of N tokens. FSCTrans randomly splits it into two sub-sequences, namely, $x_a = (x_1, \dots, x_M)$ and $x_b = (x_{M+1}, \dots, x_N)$. Following the T5 model (Raffel et al., 2020), FSCTrans takes x_a as input and generates x_b. We train the model using the cross-entropy loss, namely, minimizing

$$\mathcal{L}_{\text{ta}} = -\sum_{i=M+1}^{N} \log\ p(x_i | x_{<i}, x_a) \tag{6}$$

The pre-training objective is similar to the code translation (i.e., code-to-code generation) task, hence encouraging CodeT5 to exploit the monolingual data more efficiently.

3.4. Inference

Finally, we perform code translation by leveraging the trained model, taking the following sub-steps:

(1) The source code of language X is tokenized and embedded (Eq. (1)).
(2) The prefix tokens are encoded into (past) hidden states by the prefix module (Eq. (2)).
(3) The code embeddings and the prefix hidden states are fed into the encoder of pre-trained CodeT5, yielding the hidden states (Eq. (3)).
(4) Start the translation process with an initial token, then continue with the following loop:
    (a) Transform the current state of the translated code into token embeddings (Eq. (1)).
    (b) The prefix module generates past hidden states for the decoder (Eq. (2)).
    (c) The decoder generates the translated code tokens using both the prefix hidden states and the encoder outputs (Eqs. (3) and (4)).
(5) Repeat Step 4 until reaching the end of the sequence or the maximum decoding length.

The procedure is summarized in Algorithm 1.

Algorithm 1 Code Translation with FSCTrans

```
Input: $x$: code in the source language; $L$: max target length
Output: $y$: translated code in the target language
 1: Tokenize $x$ and transform into token embeddings $\mathbf{e}(x)$ using Eq. (1).
 2: Calculate the prefix (past) hidden states $\mathbf{H}^p$ for the encoder using Eq. (2).
 3: Input $\mathbf{e}(x)$ and $\mathbf{H}^p_{1:K}$ into the encoder and get the encoder hidden states $\mathbf{H}$
    using Eq. (3).
 4: $y = [\langle\text{SOS}\rangle]$.                    ▷ Initialize the translated sequence
 5: while not done do
 6:    Transform $y$ into token embeddings $\mathbf{e}(y)$ using Eq. (1).
 7:    Calculate the prefix (past) hidden states $\mathbf{S}^p$ for the decoder using Eq. (2).
 8:    Input $\mathbf{e}(y_{t-1})$ and $\mathbf{S}^p_{1:K}$ into CodeT5 decoder and generate the next token
       $y_t$ using Eq. (4).
 9:    if $t \neq \langle\text{EOS}\rangle$ and $|y| \leq L$ then
10:       $y = y \oplus y_t$                                  ▷ Append $y_t$ to $y$
11:    else
12:       return $y$
13:    end if
14: end while
```

4. Experimental setup

We build a few-shot code translation dataset and evaluate the performance of FSCTrans on the dataset. We further explore the effect of task adaptation, prompt learning, and different hyperparameters. In specific, we address the following research questions:

- **RQ1: How effective is FSCTrans in few-shot code translation?**
  We evaluate the effectiveness of FSCTrans in few-shot code translation on four language pairs, namely, Java→Python, Python→Java, Java→C#, and C#→Java#. We compare FSCTrans with the state-of-the-art code translation approaches such as CodeT5 and TransCoder.
- **RQ2: How effective is FSCTrans under different sizes of training data?**
  A fundamental question for few-shot learning is how much data is considered enough. We vary the size of the training set and evaluate the performance of FSCTrans on the Java↔Python translation pair since it has sufficient parallel data for training.
- **RQ3: How does the task-adapted prompt learning impact the performance of FSCTrans?**
  We conduct an ablation study to verify the effect of prompt learning and task adaptation in FSCTrans.
- **RQ4: How different hyperparameters affect the performance of FSCTrans?**
  Finally, to investigate the effect of different hyperparameters, we evaluate FSCTrans under different batch sizes and learning rates.

4.1. Datasets

Data for Task Adaptation. In the task adaptation step, we continually pre-train CodeT5 using the same monolingual data as CodeT5, which is from CodeSearchNet (Husain et al., 2019) and Github. Note that this step will not introduce extra data to train FSCTrans since the data used for task adaptation is the same to that used in the pre-training phase. The statistics of this monolingual dataset are summarized in Table 1. CodeSearchNet consists of function-level code of six programming languages, among which we choose Java and Python, two of the most popular languages. The original data is in the JSON format with fields such as source code, docstring, and code tokens. All comments are removed from the code. We also collect C# code snippets from public repositories in Github following CodeT5 for building the Java↔C# pairs, which extends the monolingual corpus of C#. For each language pair, we mix lines of code tokens of the two corresponding languages and randomly split them for training the task-adaptation objective.

Table 1
Statistics of the monolingual data for task adaption.

	Java	C#	Python
# functions	453,657	104,656	411,920

Table 2
Statistics of the parallel data for few-shot code translation.

Translation Pair	Train	Valid	Test	AVG Length*
Java ↔ Python	8	8	1,402	118
Java ↔ C#	8	8	1,000	50

*AVG Length stands for the average number of tokens per code snippet after tokenization.

Data for Prompt Learning and Translation. We build a few-shot dataset by selecting two language pairs that have adequate parallel data for testing: (1) Java↔Python, which consists of parallel solutions of online judge problems (GeeksforGeeks, 2023) sampled from the test set of TransCoder (Rozière et al., 2020); (2) Java↔C#, which consists of parallel functions of real-world projects sampled from the CodeXGLUE code translation dataset (Lu et al., 2021). The statistics of this parallel dataset are summarized in Table 2.

Following (Perez et al., 2021), we split our few-shot dataset into the training, validation, and test sets. To simulate the example-based in-context learning in large language models, we select only 8 demonstration examples for training and validation respectively. The remaining data is used as test sets to mitigate bias in evaluation. Since the remaining set for Java↔C# is too large, we further sample 1,000 random pairs from it for testing.

4.2. Baselines

Since FSCTrans is the first work on few-shot code translation, we compare FSCTrans with a variety of SOTA code translation approaches, including Transformer (Vaswani et al., 2017), SOTA approaches that directly fine-tune code PLMs such as CodeBERT (Feng et al., 2020) and CodeT5 (Wang et al., 2021), SOTA unsupervised approaches such as TransCoder (Rozière et al., 2020), a prompt tuning based approach SoftPrompt (Wang et al., 2022), a low-rank adaption based approach LoRA (Hu et al., 2022b). We also consider the simple copy as a baseline. Overall, we compare FSCTrans with the following approaches:

(1) **Simple Copy**: an approach that directly copies the source sequence as the translated sequence. This baseline can be used as a sanity check in translation tasks.
(2) **Transformer** (Vaswani et al., 2017): a sequence-to-sequence model that is widely used for translation. We implement the model based on the HuggingFace Transformer. We use the tokenizer of CodeBERT but retrain the entire model from scratch on our few-shot dataset. This baseline is a typical non-pretraining approach.
(3) **CodeBERT** (Microsoft, 2022): a code translation approach based on CodeBERT which is implemented by CodeXGLUE (Lu et al., 2021). This approach takes CodeBERT as an encoder and trains a Transformer decoder from scratch. We use the same tokenizer and token embeddings as in CodeBERT. We fine-tune this model using the same hyperparameters as in CodeXGLUE.
(4) **CodeT5** (Salesforce, 2023): a code translation approach based on CodeT5. CodeT5 adopts a sequence-to-sequence architecture and can be naturally adapted to code translation. We fine-tune CodeT5 using the same hyperparameters and data as in our prompt learning stage.
(5) **TransCoder** (Rozière et al., 2020): an unsupervised code translation method based on denoising auto-encoder (DAE) (Vincent et al., 2008) and back-translation (BT) (Lample et al., 2018). Although TransCoder does not require the availability of parallel code, it requires an enormous amount (e.g., 739M functions) of monolingual code for training. To facilitate the application of TransCoder to Java↔C# translation pairs, we take CodeT5 as the cross-lingual language model. For a fair comparison, we train it with the same amount of monolingual data used in FSCTrans. We follow the same experimental setup as in the original paper.
(6) **SoftPrompt** (Wang et al., 2022): a soft prompt-tuning approach based on CodeT5. SoftPrompt directly inserts trainable prompt tokens into the input of CodeT5 and optimizes the prompt embedding together with CodeT5. SoftPrompt has been shown to outperform fine-tuning in full-data code translation. Hence, we are more interested in how it performs in few-shot code translation. We use the same hyper-parameter settings as in the original paper.
(7) **LoRA** (Hu et al., 2022b): a parameter-efficient tuning method to adapt LLMs based on low-rank adaption. LoRA decomposes the parameter change of the dense layers into low-rank matrices. During optimization, it freezes the pre-trained weights and merely updates the low-rank matrices. To compare with our approach, we apply LoRA to CodeT5 using the same hyperparameters and training data as the CodeT5 baseline. We follow the same experimental setup of LoRA as in the original paper.

To ensure fair comparisons, all PLM-based baselines are pre-trained on the same 8.35M instances as FSCTrans. For example, both CodeBERT and CodeT5 are pre-trained on CodeSearchNet. FSCTrans and other baselines such as TransCoder, SoftPrompt, and LoRA are all based on CodeT5, which means they are pre-trained on the same CodeT5 dataset. Therefore, the pre-training stages of all baselines used the same scale of data as FSCTrans's task adaptation stage. Transformer is a typical non-pretraining approach, so it is inapplicable to be pre-trained on large-scale monolingual data. Therefore, we train it on the same parallel data as we fine-tune FSCTrans.

4.3. Metrics

We evaluate all approaches using three popular metrics, namely, **BLEU** (Papineni et al., 2002), **CodeBLEU** (Ren et al., 2020), and **Exact Match (EM)** (Lu et al., 2021). BLEU calculates the ratio of n-grams that match references in the target language. We use the BLEU-4 score which calculates the weighted average of BLEU scores with 1–4 grams. CodeBLEU is a metric that is specifically designed for code synthesis. It is calculated as a weighted combination of the original BLEU, the weighted n-gram match, the syntactic AST match, and the semantic data-flow match. We parse the translated code using Tree-sitter (Tree-sitter, 2023) which is robust to syntax errors in the results (Chada and Natarajan, 2021). EM measures the percentage of translations that exactly match the reference code. We calculate BLEU-4, CodeBLEU, and EM using the same scripts in CodeT5. During our experiments, we observe a phenomenon where multiple tokens may be treated as a single token by the evaluator, which causes incorrect BLEU scores. For example, `List.sort()` would be treated as a single token in BLEU calculation though involving five tokens. To tackle this problem, we post-process all translations by inserting spaces between all tokens. For instance, "`List.sort()`" is stretched into "`List . sort ()`".

4.4. Implementation details

Our model is implemented based on the CodeT5-base model from the open-source collection of HuggingFace Transformer (Salesforce, 2021). We continually pre-train CodeT5 using the *T5ForConditional Generation* class. We implement prefix modules by passing the prefix hidden states (i.e., values of the *past_key_values* parameter of Transformer) to the activation layers of CodeT5. All models use the default tokenizer and vocabulary of CodeT5. The batch size for all models is

Table 3
Performance of few-shot code translation.

Approach	Java→Python			Python→Java			Java→C#			C#→Java		
	BLEU-4	EM	CodeBLEU	BLEU-4	EM	CodeBLEU	BLEU-4	EM	CodeBLEU	BLEU-4	EM	CodeBLEU
Simple Copy	33.20	0.00	–	32.12	0.00	–	18.69	0.00	–	18.55	0.00	–
Transformer	18.70	0.00	22.74	20.62	0.00	29.13	9.39	0.00	32.54	6.50	0.00	24.22
CodeBERT	18.70	0.00	22.74	20.62	0.00	29.12	9.39	0.00	32.28	6.50	0.00	24.20
CodeT5	50.89	0.64	43.74	37.50	0.00	41.36	38.69	**26.40**	51.44	25.74	6.30	40.22
TransCoder	13.89	0.00	32.13	15.74	0.00	42.55	25.05	0.40	39.03	12.63	0.00	44.54
SoftPrompt+CodeT5	41.38	0.50	37.59	37.24	0.14	41.47	36.53	19.20	50.16	31.20	8.00	44.39
LoRA+CodeT5	45.16	0.86	40.41	46.32	0.57	48.46	62.29	14.8	**65.53**	**53.28**	13.1	57.06
FSCTrans	**60.58**	**4.64**	**51.85**	**54.76**	**1.50**	**53.20**	**63.09**	22.50	65.12	48.99	**25.80**	**61.37**
- w/o TA	42.73	0.50	37.93	40.58	0.00	43.22	35.02	18.50	45.12	24.00	7.90	35.43
- w/o PTL	53.89	1.64	45.61	49.41	0.21	48.69	43.23	19.50	54.15	38.42	26.00	53.46

* The parallel training data size for prompt learning and fine-tuning is 8. TA = task adaptation; PTL = prompt learning.

set to 12. We set the prefix token length to 100. The hidden size of the fully-connected prefix encoder is set to 512. The max lengths for the input and output sequences are set to 320 and 256, respectively.

We train all models on a Linux machine with Ubuntu 20.04.2 and a GPU of Nvidia RTX 3090. All models are optimized using the AdamW with an initial learning rate of 1e-5. We use the same linear learning rate scheduler with 1,000 warm-up steps. In the prompt learning step, we use the same training strategy as CodeT5 such as early stop. The training process stops if the BLEU score and the loss in the validation set do not improve for a certain number (i.e., patience) of consecutive epochs. We set patience to 300 for fine-tuning baselines, SoftPrompt baselines, and prompt learning of FSCTrans. In the task adaptation step, we continually re-train the CodeT5 checkpoint for around 100K steps. In the code translation step, we use beam search as the decoding strategy with a beam size of 10.

5. Results

5.1. Effectiveness in few-shot code translation (RQ1)

Table 3 compares the performance of FSCTrans against baseline approaches on four few-shot code pairs, each given only 8 examples. Broadly, FSCTrans achieves the best performance across all language pairs.

Compared to the state-of-the-art PLM, CodeT5, FSCTrans is superior with a remarkable margin. For example, on the C#→Java translation, FSCTrans gains a BLEU score of 48.99, which is 90.33% greater than that of CodeT5 (BLEU-4=25.74); FSCTrans also improves CodeT5 by 309.52% in terms of EM. Such improvement is consistent across most of the translations. On average, FSCTrans improves CodeT5 by 54.61% and 31.59% in terms of BLEU-4 and CodeBLEU, respectively. This indicates that by task-adapted prompt learning, FSCTrans can adapt pre-trained PLM to the code translation task effectively even with a few examples.

FSCTrans significantly outperforms TransCoder, the state-of-the-art unsupervised code translation method. By demonstrating the model with only 8 parallel examples, FSCTrans gains 255.95% and 47.76% improvement in terms of BLEU-4 and CodeBLEU. We have observed that unsupervised approaches are hungry on monolingual data though they do not require parallel code. With the same amount of monolingual data, FSCTrans can adapt the original CodeT5 to code translation more effectively than unsupervised tasks such as denoising auto-encoder and back-translation. We also notice that TransCoder gains lower BLEU scores, but relatively high CodeBLEU scores that are comparable to CodeT5. This is consistent with the findings by Rozière et al. (2020) that BLEU could not correlate well with the translation performance of TransCoder.

FSCTrans demonstrates a considerable strength over SoftPrompt, a prompt tuning based code translation method. On average, FSCTrans improves SoftPrompt by 55.79% and 33.57% in terms of BLEU-4 and CodeBLEU. Though SoftPrompt demonstrates its effectiveness in full-data code translation (Wang et al., 2022), it shows worse performance than the original CodeT5 in few-shot code translation. The results suggest that directly employing prompt tuning does not resolve the low-resource challenge of few-shot code translation. Explicit task adaptation is important to mitigate the gap between pre-training objectives and the code translation task. We will discuss more details about this issue in Sections 5.3 and 6.1.

FSCTrans also outperforms LoRA, the state-of-the-art parameter-efficient tuning approach. FSCTrans improves LoRA by 11.40% and 11.25% in terms of BLEU-4 and CodeBLEU, respectively, demonstrating the superb efficacy of FSCTrans in few-shot code translation when most of the model parameters are frozen. The results imply that combining parameter-efficient tuning with task adaptation is more effective than utilizing parameter-efficient tuning alone, particularly in low-resource scenarios.

It is worth noting that Transformer and CodeBERT perform worse than simply copying source tokens in this experimental setting. We manually inspect the translation and find that they both severely overfit our few-shot dataset. For example, they simply remember the 8 examples and randomly select one as the translation. This is because adapting CodeBERT (a Transformer encoder) to code translation (a sequence-to-sequence problem) requires us to train a Transformer decoder with 47M parameters from scratch with merely 8 examples, which is a data-hungry process. The Transformer also requires optimizing 172M randomly initialized parameters.

In contrast, CodeT5 is based on a sequence-to-sequence framework, and its parameters are pre-trained without the need for training from scratch. Hence, CodeT5 has less risk of overfitting than CodeBERT and Transformer. Simply copying source tokens results in good performance, presumably because different programming languages often overlap in terms of vocabulary, keywords, and operations, since they are developed in English. This enables the simple copy method to generate translations with highly matching n-grams, yielding high BLEU scores.

We also notice that CodeT5, SoftPrompt and FSCTrans achieve low EM scores on the Java↔Python translation. This is probably because the Java↔Python pairs have longer code sequences than Java↔C# (as Table 2 shows), which makes it exponentially difficult to translate the exact reference. This suggests that EM metric may not be suitable for assessing long translations. Besides, we have observed that in baseline models such as simple copy, Transformer, and CodeBERT, the EM score on the task is zero. For these earlier models, the EM metric is overly strict, making it quite challenging to achieve a translation result that is exactly the same as the reference.

5.2. Effect of training data size (RQ2)

Fig. 3 shows the performance of FSCTrans and CodeT5 on the Java↔Python dataset under different data sizes. We vary the size of the training set from 0 to full data. When training size is greater than

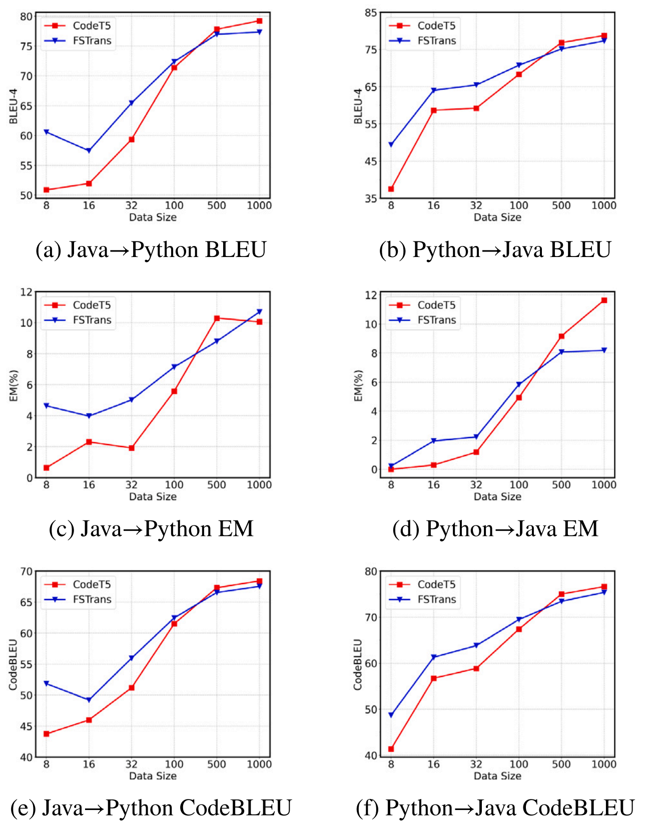

Fig. 3. Performance of FSCTrans and CodeT5 under different training data sizes.

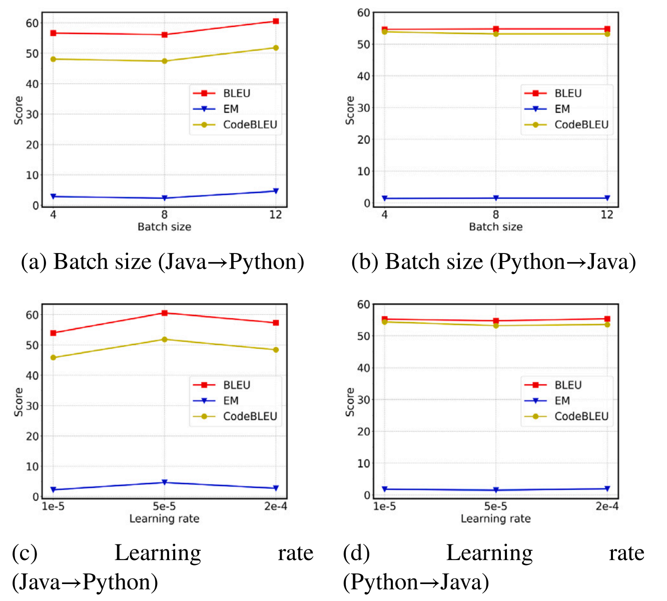

Fig. 4. Performance of FSCTrans under different hyperparameters in Java↔Python translation.

100, we keep the size of validation set to 100 to leave enough instances for testing.

Overall, FSCTrans outperforms CodeT5 when a small number (<100) of examples are provided. The smaller the data size is, the more significant the improvement becomes. For example, FSCTrans improves the BLEU score by 19.04% under only 8 examples. The improvement decreases to 1.4% when the data size is equal to 100. FSCTrans slightly underperforms CodeT5 under big data sizes (e.g., $\geq$ 500). Nevertheless, the performance is competitive given that FSCTrans only needs 8.9% trainable parameters with respect to CodeT5.

The results suggest that by only updating parameters in the prefix module, prompt learning method can effectively adapt the pre-trained model to scarce data while retaining the pre-trained knowledge in CodeT5.

5.3. *Impact of task-adapted prompt learning (RQ3)*

To assess the impact of task-adapted prompt learning, we compare FSCTrans with two variants that exclude the task adaptation (w/o TA) and prompt learning (w/o PTL) steps respectively. For the latter comparison, we replace prompt learning with fine-tuning as in CodeT5. The results are shown in Table 3.

Task adaption, by mitigating the gap between pre-training objectives and code translation, is effective in few-shot code translation. As we can see, compared with fine-tuning (i.e., CodeT5), task adaptation brings an average of 22.96% improvement in terms of BLEU. The improvement is more significant when prompt learning is also applied. Through task adaptation, FSCTrans gains 87.08% improvement in terms of BLEU in the C#→Java translation and 34.29% improvement on average compared with FSCTrans without task adaptation.

Similarly, the prompt learning step in FSCTrans also demonstrates its effectiveness when task adaptation is applied. FSCTrans outperforms the variant without prompt learning on all translation pairs, with an average of 12.71% improvement in terms of BLEU and 8.85% in terms of CodeBLEU. We also observe that simply applying prompt learning without task adaption (FSCTrans - w/o TA) decreases the BLEU score by 7.6% on average compared to the origin CodeT5. This indicates that the task adaption step is crucial to the prompt learning approach.

5.4. *Impact of hyperparameters (RQ4)*

We evaluate the performance of FSCTrans under different batch sizes and learning rates in the Java↔Python translation.

Figs. 4(a) and 4(b) show the impact of different batch sizes. We vary the batch size to 4, 8, and 12, respectively. We observe that larger batch sizes (e.g., 12) boost the performance in the Java→Python translation, while having slight effect on the Python→Java translation.

Figs. 4(c) and 4(d) show the impact of different learning rates. We vary the learning rate to $1e-5$, $5e-5$, and $2e-4$ respectively. As we can see, FSCTrans with default learning rate (i.e., $5e-5$) outperforms others in the Java→Python translation, while it has a little impact on the Python→Java translation.

5.5. *Qualitative analysis*

Here we provide concrete examples of code translation to further investigate the capability of FSCTrans in few-shot code translation.

Effectiveness Analysis. Fig. 5 shows an example of Python→Java translation for a problem named "productSubSeqCount". This translation requires to map the keywords (e.g., len→size(), and→&&) and grammar rules (e.g., brackets and loop structures) between two languages. In general, both CodeT5 and FSCTrans correctly translate most of the keywords and grammar rules, probably due to the cross-lingual pre-training objectives of CodeT5. We notice that CodeT5 (Fig. 5(c)) fails to generate a coherent translation. It does not follow the main structure of the original code and only keeps a few simple tokens, yielding translations with totally different semantics. By comparison, the translation by FSCTrans (Fig. 5(d)) is more relevant to the ground

```
def productSubSeqCount (arr , k):
    n = len(arr)
    dp = [[0 for i in range (n+1)]
        for j in range (k+1)]
    for i in range (1,k+1) :
        for j in range (1,n+1) :
            dp[i][j] = dp[i][j-1]
            if arr[j-1]<=i and arr[j-1]>0 :
                dp[i][j]+=dp[i//arr[j-1]][j-1]+1
    return dp[k][n]
```

(a) The original code of Python

```
public static int productSubSeqCount
        (ArrayList<Integer>arr,int k){
    int n = arr.size();
    int dp[][]=new int[k+1][n+1];
    for (int i=1; i<=k; i++){
        for (int j=1; j<=n; j++){
            dp[i][j] = dp[i][j-1];
            if(arr.get(j-1)<=i&&arr.get(j-1)>0)
                dp[i][j] +=
                    dp[i/arr.get(j-1)][j-1]+1;
        }
    }
    return dp[k][n];
}
```

(b) The ground truth translation of Java

```
static int productSubSeqCount (int arr[],int k){
    int n = arr.length ;
    int dp[][] = new int[n+1];
    for (int i=1; i<=k; i++) {
        for (int j=1; j<=n; j++) {
            if (arr[j-1]<=i)
                dp [i][j] += dp[i][j-1]+1;
            if (arr[j-1]>0)
                return dp [k][n];
        }
        return dp[k][n][k-1];
    }
    return dp[k][n];
}
for(int i=1;i<=k;i++){for(int j=1;j<=j;j++){
        ...
```

(c) Code translation by CodeT5

```
static int productSubSeqCount
        (int arr[], int k){
    int n = arr.length;
    int dp[][] = new int [n+1];
    for (int i=1; i<n; i++){
        for (int j=1; j<k; j++){
            dp[i][j] = dp[i][j-1];
            if(arr[j-1]<=i && arr[j-1]>0)
                dp[i][j] +=
                    dp[i//arr[j-1]][j-1]+1;
        }
    }
    return dp[k][n];
}
```

(d) Code translation by FSCTrans

Fig. 5. Python→Java translations for the "productSubSeqCount" program by CodeT5 and FSCTrans. For ease of rendering, we omit marginal parts in CodeT5 translation.

```
static int maxvolume (int s){
    int maxvalue = 0;
    for (int i = 1; i <= s - 2; i++){
        for (int j = 1; j <= s - 1; j ++){
            int k = s - i - j;
            maxvalue = Math.max(maxvalue , i*j*k);
        }
    }
    return maxvalue;
}
```

(a) The original Java code

```
def maxvolume ( s ) :
    maxvalue = 0
    i = 1
    for i in range ( s - 1 ) :
        j = 1
        for j in range (s):
            k = s - i - j
            maxvalue = max(maxvalue , i*j*k)
return maxvalue
```

(b) The ground truth translation of Python

```
def maxvolume ( s ) :
    maxvalue = 0
    for i in range ( 1 , s - 2 ) :
        for j in range ( 1 , s - 1 ) :
            k = s - i - j
            maxvalue = max ( maxvalue , i * j * k )
return maxvalue
```

(c) Code translation by CodeT5

```
def maxvolume ( s ) :
    maxvalue = 0
    for i in range ( 1 , s ) :
        for j in range ( 1 , s - 2 ) :
            for k in range ( 1 , s - 1 ) :
                maxvalue = max ( maxvalue , i * j * k )
return maxvalue
```

(d) Code translation by FSCTrans

Fig. 6. Java→Python translations for the "maxvolume" program by CodeT5 and FSCTrans.

truth. Despite minor misplacement of loop conditions, it keeps the main semantics of the input code. This indicates that FSCTrans can understand the goal of the translation task (i.e., mapping keywords and grammars while retaining the semantics) in the few-shot scenario better than CodeT5.

This example demonstrates the superiority of FSCTrans in few-shot code translation, affirming its strong ability to rapidly transfer PLMs to the translation task.

Error Analysis. Although FSCTrans achieves the new state-of-the-art, it might occasionally produce errors or worse translations. Fig. 6 shows a Java→Python translation for a "maxvolume" problem. As can be seen, FSCTrans fails to translate the expression $k = s\text{-}i\text{-}j$ while CodeT5 translates it correctly. Moreover, both approaches fail to translate the loop conditions. This indicates that translating complicated conditions is challenging in the few-shot scenario. In future work, we will conduct empirical research on the error types, and improve our model for these challenging translations.

6. Discussion

6.1. How can the task-adapted prompt learning enhance code translation in the few-shot setting?

We believe that prompt learning is able to transfer the knowledge to the downstream task more effectively in the few-shot scenario. Unlike fine-tuning which updates the entire PLM, prompt learning only updates the prefix module which involves only 8.9% number of parameters. This significantly mitigates the risk of overfitting compared with fine-tuning.

Furthermore, the task adaptation step can bridge the gap between pre-training objectives and the downstream translation task, thus enables more efficient prompt learning. The original CodeT5 is trained with MSP, IT, MIP, and BDG tasks on both unimodal (PL-only) and bimodal (PL-NL) data. Though the bimodal generation task may endow CodeT5 with the ability of PL-NL alignment, none of them supports PL-to-PL training as the downstream code translation task does. This discrepancy is even harder to overcome when the examples are scarce. With the task adaptation objective, CodeT5 can adjust its knowledge

Table 4
Comparison of ChatGPT and FSCTrans.

Approach	Java→Python			Python→Java			Java→C#			C#→Java		
	BLEU-4	EM	CodeBLEU	BLEU-4	EM	CodeBLEU	BLEU-4	EM	CodeBLEU	BLEU-4	EM	CodeBLEU
ChatGPT(GPT3.5)	65.45	2.00	**59.33**	76.29	**3.00**	75.71	55.14	8.00	55.02	71.02	32.00	77.82
ChatGPT(GPT4)	**66.25**	3.00	57.49	**78.52**	2.00	**76.30**	56.07	15.00	55.21	**81.06**	**49.00**	**79.87**
FSCTrans	60.58	**4.64**	51.85	54.76	1.5	53.20	**63.09**	**22.50**	**65.12**	48.99	25.80	61.37

with real translation examples and learn the naturalness of complete code snippets, which eventually benefits both prompt learning and fine-tuning.

6.2. Comparison with LLMs

Given that LLMs also exhibit exceptional proficiency in few-shot learning, we further investigate how the efficacy of FSCTrans aligns with that of LLMs in the context of code translation tasks. We compare FSCTrans with ChatGPT(GPT3.5 and GPT4) (OpenAI, 2023), a state-of-the-art LLM that has gained widespread popularity for its remarkable performance in various domains. We randomly sample translation pairs from TransCoder (Java↔Python) and CodeXGLEU (Java↔C#) as the test sets, conforming to the FSCTrans experiment. In our experiments, we employ the standard prompting pipeline for ChatGPT by appending multiple input–output examples to the input. For Java↔C# translations, we prompt ChatGPT(GPT3.5 and GPT4) with 8 parallel code pairs to be consistent with other models. For Java↔Python translations, however, it is infeasible to utilize the prompt of 8 examples due to the greater code length and the inherent restriction of 2048 tokens by the free version ChatGPT(GPT3.5). Through extensive testing, we choose to use a prompt of 2 translation examples in the tests of ChatGPT(GPT3.5), which comes with the optimal result. In the experiments of ChatGPT(GPT4), we prompt it with 8 parallel code pairs to be consistent with FSCTrans.

Table 4 presents the comparison results. Overall, ChatGPT(GPT4) demonstrates the most favorable results due to the ultra-large parameters and data for training, slightly outperforming ChatGPT(GPT3.5). We notably see that FSCTrans exhibited superior performance in Java→C# translation, possibly because ChatGPT's training set involves smaller C# code compared to Java and Python. The results also imply that FSCTrans is more effective when the training data size is small.

6.3. Threats to validity

We have identified the following threats to our approach:

Evaluations in domain-specific languages. We evaluate FSCTrans on four translations in the pre-training corpus of CodeT5, including Java→Python, Python→Java, Java→C#, and C#→Java. All of them are popular languages. Thus, they may not represent domain-specific languages such as Solidity. In future work, we will evaluate FSCTrans on more PL pairs that are not included in the CodeT5 corpus.

Generalization to other pre-trained programming language models. Our model is merely built upon CodeT5. Although CodeT5 is one of the most popular PLMs for code and has demonstrated promising results in code translation, other code PLMs such as PLBART (Ahmad et al., 2021) and Codex (Chen et al., 2021) may have different performances. We leave few-shot code translation on other code PLMs for our future work.

7. Related work

7.1. Code translation

Code translation has been a well-studied research area in software engineering (Mossienko, 2003; Karaivanov et al., 2014; Chen et al., 2018; Rozière et al., 2020; Wang et al., 2021). A common way of code translation is to treat source code as plain texts and apply statistical machine translation (SMT) techniques (Nguyen et al., 2013; Karaivanov et al., 2014; Aggarwal et al., 2015). For example, Nguyen et al. (2013) applied the phrase-based SMT model to map phrases of code tokens between source and target languages. Karaivanov et al. (2014) extended the phrase-based approach with grammars and pruned the search space of machine translation. Aggarwal et al. (2015) built an SMT model to migrate code from Python 2 to Python 3.

Recently, inspired by the great success of pre-trained models in NLP, a boom arises in pre-trained models on code migration. Chen et al. (2018) made the first attempt to translate source code by leveraging a tree-based encoder–decoder model. Rozière et al. (2020) presented an unsupervised code translation model with the attention mechanism, and adapted the model to the target language using back-translation in a weakly-supervised way. Chen and Abedjan (2021) proposed a retrieval-based method to search translations from a big code database directly. Their method measures cross-lingual code similarities by mapping multilingual source code to a unified representation. Ahmad et al. (2021) pre-trained the popular PLM, BART (Lewis et al., 2020) in programming languages. They found that PLM generates significantly more syntactically correct code translations than traditional methods.

The main limitation of SMT and deep learning based methods lies in the dependency on large parallel code corpus. They both alleviate this problem by focusing only on popular programming languages (e.g., Java↔C#) that have plenty of open-source projects. Although the retrieval-based methods do not require a parallel dataset, they can only search translations that already exist in the codebase. Unsupervised methods (Rozière et al., 2020, 2022; Szafraniec et al., 2022; Liu et al., 2023a; Ahmad et al., 2023) do not need the availability of parallel data. However, they rely on a huge amount of mono-lingual code (e.g., 739M functions in TransCoder) for training a cross-lingual language model. By contrast, FSCTrans needs far less monolingual code (e.g., 1.2M functions for task adaptation and 8.35M functions for pre-training).

More recently, prompt-tuning has become a promising alternative to the data-hungry fine-tuning method. Wang et al. (2022) found that soft prompt tuning outperforms CodeT5 in full-data code translation, while underperforming in few-shot code translation. ChatGPT (Jiao et al., 2023a) applies in-context learning to trigger its translation ability through a set of carefully curated prompts by humans. Unlike soft prompt and ChatGPT which directly employ prompt learning, FSCTrans leverages task adaption to bridge the gap between PLMs and the translation task, and thus achieves superior performance in few-shot code translation.

Overall, differing significantly from these existing methods, FSCTrans focuses on the supervised way for few-shot code translation. FSCTrans extends code PLM with task-adapted prompt learning techniques to tackle the few-shot problem. It generalizes the knowledge learned from a small number of examples to unseen code snippets and hence works on programming languages with little parallel data.

7.2. Pre-trained language models for code

Recently, PLMs have achieved great success in programming languages (Feng et al., 2020; Guo et al., 2021; Ahmad et al., 2021; Mastropaolo et al., 2021; Wang et al., 2021). Feng et al. (2020) made a pioneer attempt to adapt pre-trained models for programming languages. Their proposed model called CodeBERT extends the BERT model with an RTD task, and is pre-trained on the CodeSearchNet dataset with six PLs. Guo et al. (2021) proposed GraphCodeBERT, a graph-based extension of BERT for code. In addition to the MLM task in BERT, the model is pre-trained with two novel tasks, namely, edge prediction and node alignment, to learn structural information from data flow graphs. Ahmad et al. (2021) introduced PLBART, a denoising sequence-to-sequence model learned from both PL and NL data. Pre-trained on Java and Python functions and their associated NL texts, PLBART shows great effectiveness on code summarization, code generation, code clone detection, and program repair. Mastropaolo et al. (2021) adapted the T5 model to programming languages. They pre-trained the T5 model on parallel data of source code and English sentences and evaluated the model on four generation tasks. Wang et al. (2021) proposed CodeT5, which extends the T5 model with an identifier-aware pre-training objective and an NL-PL dual-generation task. It achieves the best performance on all fourteen code-related tasks in CodeXGLUE (Lu et al., 2021). Codex (Chen et al., 2021) is a large language model developed by OpenAI. It is GPT-3 language model with up to 12 billion parameters which has been fine-tuned with 159 GB of code examples from GitHub repositories.

PLMs can learn cross-lingual knowledge by pre-training on large code corpora. Though they can be fine-tuned on the code translation task, the performance is often sub-optimal due to the gap between pre-training objectives and the code translation task. Instead of fine-tuning PLMs directly, FSCTrans employs a novel prompt learning method to adapt the PLM to a few translation examples without updating all parameters. Furthermore, FSCTrans mitigates the gap between pre-training and translation objectives through task adaptation.

8. Conclusion

FSCTrans is a novel approach for low-resource code translation. To efficiently reuse pre-trained knowledge in the PLM, FSCTrans employs prompt learning, adjusting the inputs to the PLM while retaining the pre-trained parameters. FSCTrans also employs a task adaptation solution, bridging the pre-training of PLM to the code translation task. The evaluation results show that FSCTrans outperforms supervised, unsupervised, and large PLM baselines by a remarkable margin in few-shot code translation.

CRediT authorship contribution statement

Xuan Li: Methodology, Validation, Writing – original draft. **Shuai Yuan:** Data curation, Investigation, Software, Writing – original draft. **Xiaodong Gu:** Conceptualization, Methodology, Writing – review & editing. **Yuting Chen:** Writing – review & editing, Investigation. **Beijun Shen:** Writing – review & editing, Funding acquisition, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Source code and dataset to reproduce this work are available online: https://github.com/yssjtu/code-translation.

Acknowledgments

This research is supported by National Key Research and Development Program of China (Grant No. 2023YFB4503802) and National Natural Science Foundation of China (Grant No. 62102244, 62032004 and 62272296).

References

Aggarwal, K., Salameh, M., Hindle, A., 2015. Using machine translation for converting *Python 2* to *Python 3* code. PeerJ Prepr. 3, e1459.

Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K., 2021. Unified pre-training for program understanding and generation. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. NAACL-HLT, pp. 2655–2668.

Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K., 2023. Summarize and generate to back-translate: Unsupervised translation of programming languages. In: Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics. EACL, pp. 1528–1542.

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al., 2020. Language models are few-shot learners. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems. NeurIPS.

Chada, R., Natarajan, P., 2021. FewshotQA: A simple framework for few-shot learning of question answering tasks using pre-trained text-to-text models. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 6081–6090.

Chen, B., Abedjan, Z., 2021. RPT: effective and efficient retrieval of program translations from big code. In: IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings. ICSE-Companion, pp. 252–253.

Chen, X., Liu, C., Song, D., 2018. Tree-to-tree neural networks for program translation. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems. NeurIPS, pp. 2552–2562.

Chen, M., Tworek, J., Jun, H., et al., 2021. Evaluating large language models trained on code. CoRR, abs/2107.03374.

Cui, N., Jiang, Y., Gu, X., Shen, B., 2022. Zero-shot program representation learning. In: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. ICPC, pp. 60–70.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M., 2020. CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics. ACL, pp. 1536–1547.

Fu, W., Menzies, T., 2017. Easy over hard: a case study on deep learning. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. ESEC/FSE, pp. 49–60.

Gao, T., Fisch, A., Chen, D., 2021. Making pre-trained language models better few-shot learners. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. ACL/IJCNLP, pp. 3816–3830.

GeeksforGeeks, 2023. Online judge problems in GeeksforGeeks platform. URL https://www.geeksforgeeks.org/.

Gu, Y., Han, X., Liu, Z., Huang, M., 2022. PPT: pre-trained prompt tuning for few-shot learning. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). ACL, pp. 8410–8423.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S.K., Clement, C.B., Drain, D., Sundaresan, N., Yin, J., Jiang, D., Zhou, M., 2021. GraphCodeBERT: Pre-training code representations with data flow. In: 9th International Conference on Learning Representations. ICLR.

Hambardzumyan, K., Khachatrian, H., May, J., 2021. WARP: Word-level adversarial reprogramming. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. ACL/IJCNLP, pp. 4921–4933.

Hu, Y., Lee, C.-H., Xie, T., Yu, T., Smith, N.A., Ostendorf, M., 2022a. In-context learning for few-shot dialogue state tracking. In: Findings of the Association for Computational Linguistics. ACL, pp. 2627–2643.

Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., 2022b. LoRA: Low-rank adaptation of large language models. In: The Tenth International Conference on Learning Representations. ICLR.

Husain, H., Wu, H., Gazit, T., Allamanis, M., Brockschmidt, M., 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. CoRR, abs/1909.09436.

Jiao, W., Wang, W., Huang, J.-t., Wang, X., Tu, Z., 2023a. Is ChatGPT a good translator? A preliminary study. arXiv preprint arXiv:2301.08745.

Jiao, M., Yu, T., Li, X., Qiu, G., Gu, X., Shen, B., 2023b. On the evaluation of neural code translation: Taxonomy and benchmark. In: 38th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 1529–1541.

Karaivanov, S., Raychev, V., Vechev, M.T., 2014. Phrase-based statistical translation of programming languages. In: Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Part of SPLASH '14. pp. 173–184.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R., 2017. Overcoming catastrophic forgetting in neural networks. Proc. Natl. Acad. Sci. 114 (13), 3521–3526.

Lample, G., Ott, M., Conneau, A., Denoyer, L., Ranzato, M., 2018. Phrase-based & neural unsupervised machine translation. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 5039–5049.

Lester, B., Al-Rfou, R., Constant, N., 2021. The power of scale for parameter-efficient prompt tuning. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 3045–3059.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., Zettlemoyer, L., 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, pp. 7871–7880.

Li, X.L., Liang, P., 2021. Prefix-tuning: Optimizing continuous prompts for generation. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. ACL/IJCNLP, pp. 4582–4597.

Liu, F., Li, J., Zhang, L., 2023a. Syntax and domain aware model for unsupervised program translation. In: 2023 IEEE/ACM 45th International Conference on Software Engineering. ICSE, pp. 755–767.

Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z., Tang, J., 2023b. GPT understands, too. AI Open http://dx.doi.org/10.1016/j.aiopen.2023.08.012.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C.B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S.K., Fu, S., Liu, S., 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In: Proceedings of NeurIPS Datasets and Benchmarks. December 2021.

Mastropaolo, A., Scalabrino, S., Cooper, N., Nader-Palacio, D., Poshyvanyk, D., Oliveto, R., Bavota, G., 2021. Studying the usage of text-to-text transfer transformer to support code-related tasks. In: 43rd IEEE/ACM International Conference on Software Engineering. ICSE, pp. 336–347.

Microsoft, 2022. CodeBERT based translation. URL https://github.com/microsoft/CodeXGLUE/tree/main/Code-Code/code-to-code-trans.

Mossienko, M., 2003. Automated cobol to java recycling. In: 7th European Conference on Software Maintenance and Reengineering. CSMR, pp. 40–50.

Nguyen, A.T., Nguyen, T.T., Nguyen, T.N., 2013. Lexical statistical machine translation for language migration. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. ESEC/FSE, pp. 651–654.

OpenAI, 2023. ChatGPT. URL https://chat.openai.com.

Papineni, K., Roukos, S., Ward, T., Zhu, W., 2002. Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. ACL, pp. 311–318.

Perez, E., Kiela, D., Cho, K., 2021. True few-shot learning with language models. In: Advances in Neural Information Processing Systems 34. NeurIPS, pp. 11054–11070.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J., 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. 21, 140:1–140:67.

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., Ma, S., 2020. CodeBLEU: a method for automatic evaluation of code synthesis. CoRR, abs/2009.10297.

Rozière, B., Lachaux, M., Chanussot, L., Lample, G., 2020. Unsupervised translation of programming languages. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems. NeurIPS.

Rozière, B., Zhang, J., Charton, F., Harman, M., Synnaeve, G., Lample, G., 2022. Leveraging automated unit tests for unsupervised code translation. In: The Tenth International Conference on Learning Representations. ICLR.

Salesforce, 2021. CodeT5-base model implementation. URL https://huggingface.co/Salesforce/codet5-base.

Salesforce, 2023. CodeT5 based translation. URL https://github.com/salesforce/CodeT5.

Shin, T., Razeghi, Y., IV, R.L.L., Wallace, E., Singh, S., 2020. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 4222–4235.

Szafraniec, M., Rozière, B., Leather, H., Charton, F., Labatut, P., Synnaeve, G., 2022. Code translation with compiler representations. CoRR, abs/2207.03578.

Tree-sitter, 2023. An incremental parsing system for programming tools. URL https://github.com/tree-sitter/tree-sitter.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I., 2017. Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems. NeurIPS, pp. 5998–6008.

Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P., 2008. Extracting and composing robust features with denoising autoencoders. In: Proceedings of the Twenty-Fifth International Conference on Machine Learning. ICML, pp. 1096–1103.

Wang, Y., Wang, W., Joty, S.R., Hoi, S.C.H., 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 8696–8708.

Wang, C., Yang, Y., Gao, C., Peng, Y., Zhang, H., Lyu, M.R., 2022. No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE, pp. 382–394.

Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M., 2020. Generalizing from a few examples: A survey on few-shot learning. ACM Comput. Surv. 53 (3), 63:1–63:34.

Zhang, B., Williams, P., Titov, I., Sennrich, R., 2020. Improving massively multilingual neural machine translation and zero-shot translation. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, pp. 1628–1639.

Xuan Li received the B.S. degree in software engineering from Shanghai Jiao Tong University, China, in 2022. She is currently pursuing her M.S. degree under the supervision of Prof. Beijun Shen. Her research interests include code translation, large language models (LLMs) and privacy protection. In 2023, Li co-authored a paper on neural code translation for the 38th ASE.

Shuai Yuan received the B.S. and M.S. degree in computer science from Shanghai Jiao Tong University, China, in 2020 and 2023, respectively. His research interests include code translation, LLMs for code and prompt learning.

Xiaodong Gu is currently an associate professor in the School of Software, Shanghai Jiao Tong University. He received a Ph.D. degree from the Department of Computer Science and Engineering from The Hong Kong University of Science and Technology, in 2017. His research interests lie in the broad areas of software engineering and deep learning, including LLMs for code, code generation, code search, and code translation. He has published over 30 papers in top conferences and journals in the field of software engineering.

Yuting Chen is an associate professor in the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He received a Ph.D. degree in computer science from Hosei University, Japan, in 2007. His research interests include program analysis and software testing, and data-driven software development. He has published over 80 papers in conferences and journals in the field of software engineering. His papers have received the Distinguished Paper Award at ICSE 2023.

Beijun Shen is an associate professor in the School of Software, Shanghai Jiao Tong University. She received a Ph.D. degree from Institute of Software, Chinese Academy of Sciences, in 2001. Her research interests include LLMs for code and code intelligence. She has published over 200 papers in top conferences and journals in the field of software engineering, and received ICSE Distinguished Paper Award in 2023.