A Joint Learning Model with Variational Interaction for Multilingual Program Translation

Anonymous Author(s)

ABSTRACT

Programs implemented in various programming languages form the foundation of software applications. To alleviate the burden of program migration and facilitate the development of software systems, automated program translation across languages has garnered significant attention. Previous approaches primarily focus on pairwise translation paradigms, learning translation between pairs of languages using bilingual parallel data. However, parallel data is difficult to collect for some language pairs, and the distribution of program semantics across languages can shift, posing challenges for pairwise program translation. In this paper, we argue that jointly learning a unified model to translate code across multiple programming languages is superior to separately learning from bilingual parallel data. We propose Variational Interaction for Multilingual Program Translation (VIM-PT), a disentanglement-based generative approach that jointly trains a unified model for multilingual program translation across multiple languages. VIM-PT disentangles code into language-shared and language-specific features, using variational inference and interaction information with a novel lower bound, then achieves program translation through conditional generation. VIM-PT demonstrates four advantages: 1) captures language-shared information more accurately from various implementations and improves the quality of multilingual program translation, 2) mines and leverages the capability of non-parallel data, 3) addresses the distribution shift of program semantics across languages, 4) and serves as a unified model, reducing deployment complexity.

CCS CONCEPTS

• **Software and its engineering** → *Automatic programming*.

KEYWORDS

Program Translation, Multi-lingual Disentanglement, Variational Interaction, Regularization

1 INTRODUCTION

Programs form the foundation of computer applications. Various programming languages have been invented to address diverse requirements. Developing a new application from scratch is time- and labor-intensive, whereas referring to and combining existing code is more efficient. Unfortunately, when combining programs written in different programming languages, the developer usually suffers from the intensive labor of manually translating programs from one programming language to another. For example, many industries spend hundreds of millions of dollars to convert code written in older programming languages (e.g., FORTRAN and COBOL) to newer ones (e.g., Java, C++) [35]. To alleviate the burden of program migration and facilitate the development of software systems, program translation, which aims to automatically translate the program from one programming language to another, has drawn significant attention in the software mining community [2, 8, 16, 21, 35, 44, 45, 48, 50, 51].

The booming development of machine learning coupled with the availability of an extensive parallel corpus of programs has led to a remarkable enhancement in the performance of program translation. For example, Nguyen et al. [26, 27] attempted to translate code across languages, leveraging phrase-based statistical machine translation and grammatical rules. Recently, due to the potent power of representation learning in the deep neural networks, most modern program translation approaches based on deep neural networks, such as sequence-to-sequence models, have advanced the state-of-the-art performance to a new level [7, 9, 13].

Nevertheless, most existing approaches primarily focus on pairwise translation, lacking exploration in multilingual program translation. Although pairwise approaches can be employed for each language pair separately, these approaches cannot leverage knowledge from languages outside the current pair, posing a challenge for learning across a pair of languages with only a few parallel data. As shown in Fig. 1, previous approaches, including the multilingual program translation approach by Zhu et al. [51], are trained only with bilingual parallel data. However, these parallel data are sparse in practical datasets. For instance, the multilingual dataset *CoST* [51] contains only 12.56% bilingual parallel data among all possible pairs. In addtion, as shown in Fig. 5 and Fig. 6. the distribution of different language pairs is imbalanced.

In this paper, we argue that jointly training a unified model to translate code across multiple programming languages is superior to separately training with pairwise data. A program implemented in different programming languages should exhibit the invariant language-shared semantics, i.e., perform the same tasks. Thus, joint training on multiple different implementations of a program can complement and enhance the learning of the language-shared representation. By constructing the unified language-shared latent space, the rich resource languages can benefit the low resource languages in the translation. However, joint learning involves a
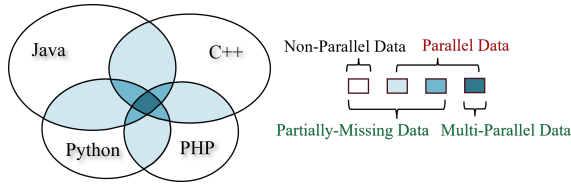
Figure 1: An example of a multilingual dataset.

new challenge: as shown in Fig. 1, practical datasets always is semi-parallel, which contain only a few complete multi-parallel samples that include all languages, while most are partially missing implementations in some programming languages. To tackle the issue of semi-parallel data, utilizing semi-supervised learning techniques is intuitive. Generative semi-supervised learning offers a solution: all data, including partially missing and multi-parallel data, can be generated from a latent distribution. All these data can be used to learn the latent distribution and generative process with an Expectation–Maximization (EM)-based algorithm [17]. Once the latent distribution and generative process are learned, program translation can be achieved with a conditional generative process.

Building upon the above idea, we propose an innovative approach called Variational Interaction for Multilingual Program Translation (VIM-PT), a disentanglement-based generative approach. VIM-PT disentangles each code into **language-shared** features and **language-specific** features based on information theory. The language-shared features should be task-specific and language-invariant like the functional semantics of the program, while the language-specific features should be language-specific and task-invariant like the grammar or syntax of the programming language. Therefore, each code can be generated through the interplay of these features. Specifically, we employ variational inference to learn the generative process, where the prior distributions of both features follow a normal Gaussian distribution. To enforce disentanglement, there will be a new variational interaction bound for the objective, compared to the Evidence Lower Bound (ELBO) in traditional variational inference, which includes three additional terms from interaction information. By optimizing this new variational interaction bound, VIM-PT learns to generate code based on disentangled features. This enables program translation from a source language to a target language by using the language-shared feature from the source language and the language-specific feature sampled from the prior distribution of the target language.

Compared to previous approaches, VIM-PT demonstrates four strengths: 1) More precisely and completely captures the language-shared feature by jointly learning from the various views of multiple implementations and improves the quality of multilingual program translation. 2) Effectively leverages the power of non-parallel and partially missing data using a generative framework. 3) Addresses the distribution shift of semantics across multiple languages. Different languages are often used for different tasks, such as Python for data science and machine learning, and JavaScript for front-end development, resulting in a distribution shift of semantics in collected data. VIM-PT addresses this by using a conditional generative framework to complete missing implementations in some

languages. 4) VIM-PT is a unified model across different translation pairs, reducing the total parameters and making deployment more convenient, especially on edge devices. To evaluate the effectiveness of the VIM-PT, an extensive experiment is conducted on the widely used dataset, which includes 7 general programming languages (i.e., C, C#, C++, Java, Javascript, PHP, and Python). It can be observed that VIM-PT performs performance gains widely over the state-of-the-art approaches.

We highlight our contributions in three key aspects:

(1) We argue that jointly training a unified model to translate code across multiple programming languages is superior to training with pairwise data separately.

(2) We propose VIM-PT, a disentanglement-based generative approach, for joint learning with variational interaction in multilingual program translation.

(3) We conducted extensive experiments to demonstrate the effectiveness of VIM-PT and the necessity of jointly learning a unified model in multilingual program translation.

2 RELATED WORKS

2.1 Program Translation

To meet some specific requirements, rule-based translation methods have been developed [5, 6, 34, 42]. Further, some works applied phrase-based statistical machine translation techniques to program translation [4, 15, 26–31], which leveraged grammatical structures of programming languages for code migration. Recently, various deep learning techniques were employed to program translations [20, 46, 47], which can be distinguished into pairwise program translation methods [7, 9, 13] and multilingual program translation methods [51] by using bilingual or multilingual corpus in the supervised learning.

The pairwise approaches trained one model for each translation direction independently [13, 23, 40]. However, a problem encountered by many existing models is that program translation datasets are usually not balanced in size for all the languages. Some languages may have much less parallel data than others. Less parallel training data can significantly affect the translation performance of low-resource languages. Therefore, some works leveraged multilingual training to improve the performance of low-resource languages, which is defined as multilingual program translation. However, the existing multilingual program translation model MuST-PT [51] is limited to the pairwise training process and does not take full advantage of the multi-parallel data and partially missing data (e.g., non-parallel data).

2.2 Exploring Non-Parallel Data for Translation

The quality of translation systems highly depends on the quality of the available parallel data. However, for most languages, parallel resources are rare or nonexistent. Some approaches have investigated the use of non-parallel data to improve existing program translation systems [12, 19, 35, 36, 41, 43]. For instance, Lachaux et al. [19] proposed DOBF that leverages the structural aspect of coding languages and pre-trains a model to recover the original version of obfuscated source code. Feng et al. [12] proposed CodeBERT, which is a Bert-like model pre-trained in an open-source GitHub repository and used in many down-stream tasks, and then much relevant

research emerged [10, 13]. Rozière et al. [35] proposed TransCoder, which trains a translation model only using the monolingual corpus with back-translation and denoising auto-encoding.

A common shortcoming of the above methods is the language-shared and language-specific representations among different languages are not disentangled in the learning. As shown in many researches, the disentanglement is crucial in such a way that every factor of variation is captured in the right part of the representation so that the partially missing data can be better utilized [11, 14, 22, 32, 37–39, 49]. For example, in neural machine translation, Zheng et al. [49] proposed the mirror-generative approach, which is a single unified architecture in which both translation models and language models share the same latent semantic space, therefore both translation directions can learn from non-parallel data. Yet the method is limited to bilingual translation and requires extra language models with more space overhead [49]. Inspired by the discussion on cross-domain disentanglement [14], we expand the variational disentanglement to the multi-domain, and it is the first attempt to explore the variational disentanglement in multi-lingual translation among programming languages.

3 METHOD

The method is discussed in this section, including the generative model by variational inference, enforcing disentanglement by interaction information, the overall framework of VIM-PT, and training the joint model from both multi-parallel samples and partially missing samples.

3.1 Generative Model via Variational Inference

Consider a sample program implemented in N languages, denoted as $(x^1, x^2, \cdots, x^N) \sim p_D(x^1, x^2, \cdots, x^N)$, where x_i represents the code written in the i-th programming language. All of these N codes exhibit the same semantics, i.e., perform the same task, while each code x_i involves the language-specific grammatical style. From the perspective of disentanglement, each code is generated through the interplay of a language-shared feature, denoted as $z^s \in Z^s$, and a language-specific feature, denoted as $z^i \in Z^i$ for the i-th programming language.
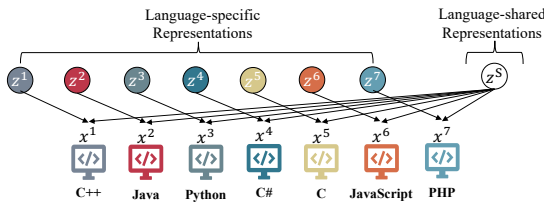

Figure 2: Generative Process.

As illustrated in Fig. 2, the generative process involves two steps: 1) The shared feature z^s is generated from the prior distribution $p_{\theta^*}(z^s)$, while the language-specific feature z^i is generated from $p_{\theta^*}(z^i)$. 2) Subsequently, the code x^i is generated from the conditional distribution $p_{\theta^*}(x^i|z^s, z^i)$. Following Kingma and Welling [18], these prior distributions $p_{\theta^*}(z^s)$ and $p_{\theta^*}(z^i)$, as well as the conditional distribution $p_{\theta^*}(x^i|z^s, z^i)$, are drawn from parametric

families of distributions $p_\theta(z^s)$, $p_\theta(z^i)$, and $p_\theta(x^i|z^s, z^i)$, respectively. Therefore, as per Kingma and Welling [18], the generative process can be formally expressed through the marginal likelihood of the joint distribution in the multilingual Bayesian network:

$$p_\theta(x^1, x^2, \cdots, x^N) = \int p_\theta(z^s)dz^s \prod_{i=1}^{N} p_\theta(z^i)p_\theta(x^i|z^i, z^s)dz^i. \tag{1}$$

While Eq. 1 is intractable, variational inference based on Variational Autoencoders (VAE) is typically employed [14, 17, 18]. Considering the inference process in variational inference, as shown in Fig. 3, the language-shared feature z^s can be jointly inferred from all codes implemented in various programming languages, while the language-specific grammatical style feature z^i is inferred from the code implemented in the i-th programming language.
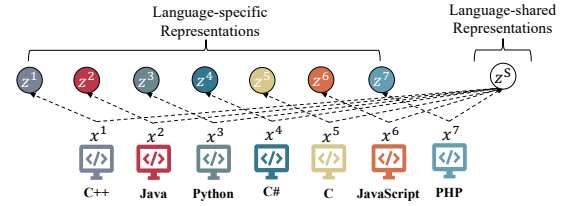

Figure 3: Inference Process.

Thus, the inference process can be formally expressed via the posterior distribution $p_\theta(z^1, z^2, \cdots, z^N, z^s|x^1, x^2, \cdots, x^N)$, which can be approximated by:

$$q_\phi(z^1, z^2, \cdots, z^N, z^s|x^1, x^2, \cdots, x^N) = \\ q_\phi(z^1|x^1)q_\phi(z^2|x^2)\cdots q_\phi(z^N|x^N)q_\phi(z^s|x^1, x^2, \cdots, x^N). \tag{2}$$

By combining Eq. 1 and Eq. 2, and omitting subscripts θ and ϕ for brevity, the log-likelihood of the joint distribution can be lower-bounded using the ELBO as:

$$\log p(x^1, x^2, \cdots, x^N)$$

$$\geq \mathbb{E}_{q(z^1, \cdots, z^N, z^s|x^1, \cdots, x^N)} \left[\log \frac{p(x^1, \cdots, x^N, z^1, \cdots, z^N, z^s)}{q(z^1, \cdots, z^N, z^s|x^1, \cdots, x^N)} \right]$$

$$= \sum_{i=1}^{N} \mathbb{E}_{q(z^i|x^i)q(z^s|x^1, \cdots, x^N)} \left[\log p(x^i|z^i, z^s) \right] \tag{3}$$

$$- \sum_{i=1}^{N} D_{KL} \left[q(z^i|x^i)\|p(z^i) \right] - D_{KL} \left[q(z^s|x^1, \cdots, x^N)\|p(z^s) \right],$$

where D_{KL} represents the KL-divergence. Specifically, in variational inference, the prior distribution $p(z^\cdot)$ is typically a Gaussian distribution, such that $p(z^\cdot) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, while the approximate posterior $q(z|x)$ can be reparameterized as $q(z|x) \sim \mathcal{N}(\mu(z|x), \sigma(z|x))$. By relying on a reparameterization trick [18], we can now jointly train all the components using gradient-based algorithms.

We formulate the generative process in Fig. 2 using Eq. 1 and the inference process in Fig. 3 using Eq. 2, then combine them to bound the log-likelihood with the ELBO in Eq. 3. However, maximizing the ELBO in Eq. 3 does not control the disagreement in inference. Disentanglement will be enforced using interaction information

from information theory, as described by Hwang et al. [14], which will be discussed in detail next.

3.2 Disentanglement by Interaction Information

In ideal disentanglement, the decomposed features should satisfy two key properties: 1) The language-shared feature $z^s \in Z^s$ and the language-specific grammatical style feature $z^i \in Z^i$ should avoid capturing redundant information. 2) For any code implemented in the i-th language, $x^i \in X^i$, the common information between x^i and others $(x^1, \cdots, x^{i-1}, x^{i+1}, \cdots, x^N)$ should be the language-shared information, i.e., $z^s \in Z^s$.

To achieve the first property, we employ the mutual information $I(\cdot; \cdot)$ between Z^s and Z^i to quantify redundancy:

$$I(Z^i; Z^s) = -I(X^i; Z^i, Z^s) + I(X^i; Z^i) + I(X^i; Z^s). \quad (4)$$

The proof of this equation can be found in Appendix A.1.

To achieve the second property, we use interaction information, a generalization of mutual information among three or more random variables, to quantify the common information among the code implemented in the i-th programming language, the codes implemented in other languages, and the language-shared features. For brevity, we denote $\{x^1, \cdots, x^{i-1}, x^{i+1}, \cdots, x^N\}$ as $\overline{x_i}$ and $\{X^1, \cdots, X^{i-1}, X^{i+1}, \cdots, X^N\}$ as $\overline{X_i}$. Formally, the common information can be expressed as:

$$I(X^i; \overline{X_i}; Z^s) = I(X^i; Z^s) - I(X^i; Z^s | \overline{X_i}), \quad (5)$$

which can be derived from the definition of the interaction information between three random variables.

Then, we combine Eq. 4 and Eq. 5 to enforce disentanglement:

$$I(X^i; \overline{X_i}; Z^s) - I(Z^i; Z^s) = -I(X^i; Z^s | \overline{X^i}) + I(X^i; Z^i, Z^s) - I(X^i; Z^i). \quad (6)$$

Unfortunately, directly maximizing Eq. 6 is still intractable. Let us analyze it term by term.

- The first term involves $q(z^s | \overline{x^i}) = \int p_D(x^i | \overline{x^i}) q(z^s | x^1, \cdots, x^N) dx^i$, where $p_D(x^i | \overline{x^i})$ is unknown. Therefore, using the variational distribution $r^i(z^s | x^i)$, the first term can be lower bounded as follows:

$$-I(X^i; Z^s | \overline{X^i}) = -\mathbb{E}_{p_D(x^1, \cdots, x^N) q(z^s | x^1, \cdots, x^N)} \left[\log \frac{q(z^s | x^1, \cdots, x^n)}{q(z^s | \overline{x^i})} \right]$$

$$= -\mathbb{E}_{p_D(x^1, \cdots, x^N) q(z^s | x^1, \cdots, x^N)} \left[\log \frac{q(z^s | x^1, \cdots, x^N) r^i(z^s | x^i)}{q(z^s | \overline{x^i}) r^i(z^s | x^i)} \right]$$

$$= -\mathbb{E}_{p_D(x^1, \cdots, x^N)} \left[D_{KL} \left[q(z^s | x^1, \cdots, x^N) \| r^i(z^s | x^i) \right] \right] \quad (7)$$

$$+ \mathbb{E}_{p_D(\overline{x^i})} \left[D_{KL} \left[q(z^s | x^i) \| r^i(z^s | x^i) \right] \right]$$

$$\geq -\mathbb{E}_{p_D(x^1, \cdots, x^N)} \left[D_{KL} \left[q(z^s | x^1, \cdots, x^N) \| r^i(z^s | x^i) \right] \right].$$

Thus, when maximizing $-I(X^i; Z^s | \overline{X^i})$ using Eq. 7, the variational distribution $r^i(z^s | x^i)$ will be learned to fit $q(z^s | x^1, \cdots, x^N)$.

- Meanwhile, the second term $I(X^i; Z^i, Z^s)$ involves $q(x^i | z^i, z^s) = \frac{q(z^i, z^s | x^i) p_D(x^i)}{\int p_D(x^i, \cdots, x^N) q(z^i, z^s | x^i, \cdots, x^N) dx^i, \cdots, dx^N}$, where $p_D(x^1, \cdots, x^N)$

and $p_D(x^i)$ are unknown. However, it can be lower-bounded using the generative distribution $p(x^i | z^i, z^s)$ as follows:

$$I(X^i; Z^i, Z^s) = \mathbb{E}_{q(z^i, z^s | x^i) p_D(x^i)} \left[\log \frac{q(x^i | z^i, z^s)}{p_D(x^i)} \right]$$

$$= H(X^i) + \mathbb{E}_{q(z^i, z^s | x^i) p_D(x^i)} \left[\log p(x^i | z^i, z^s) \right]$$

$$+ \mathbb{E}_{q(z^i, z^s)} \left[D_{KL} \left[q(x^i | z^i, z^s) \| p(x^i | z^i, z^s) \right] \right] \quad (8)$$

$$\geq H(X^i) + \mathbb{E}_{q(z^i, z^s | x^i) p_D(x^i)} \left[\log p(x^i | z^i, z^s) \right]$$

$$= H(X^i) + \mathbb{E}_{p_D(x^i, \cdots, x^N) q(z^i | x^i) q(z^s | x^i, \cdots, x^N)} \left[\log p(x^i | z^i, z^s) \right]$$

where $H(\cdot)$ represents the Shannon entropy.

- The third term $-I(X^i; Z^i)$ involves $q(z^i) = \int p_{\mathcal{D}}(x^i) q(z^s | x^i) dx$, where $p_{\mathcal{D}}(x^1)$ is unknown. We employ the Variational Information Bottleneck (VIB) [3] to lower bound it:

$$-I(X^i; Z^i) = -\mathbb{E}_{p_D(x^i)} \left[D_{KL} \left[q(z^i | x^i) \| q(z^i) \right] \right]$$

$$\geq -\mathbb{E}_{p_D(x^i)} \left[D_{KL} \left[q(z^i | x^i) \| p(z^i) \right] \right] \quad (9)$$

Here we use $-\mathbb{E}_{p_D(x^i)} \left[D_{KL} \left[q(z^i | x^i) \| p(z^i) \right] \right]$ as its lower bound with the generative distribution $p(z^i)$ defined as the standard Gaussian.

Subsequently, we can derive a lower bound for Eq. 6 by combining the derived equations. Therefore, we can enforce disentanglement across multiple programming languages by maximizing:

$$\sum_{i=1}^{N} \left[I(X^i; \overline{X^i}; Z^s) - I(Z^i; Z^s) \right]$$

$$= \sum_{i=1}^{N} \left[-I(X^i; Z^s | \overline{X^i}) + I(X^i; Z^i, Z^s) - I(X^i; Z^i) \right]$$

$$\geq \mathbb{E}_{p_D(x^1, \cdots, x^N)} \sum_{i=1}^{N} \mathbb{E}_{q(z^i | x^i) q(z^s | x^1, \cdots, x^N)} \left[\log p(x^i | z^i, z^s) \right]$$

$$- \mathbb{E}_{p_D(x^1, \cdots, x^N)} \sum_{i=1}^{N} D_{KL} \left[q(z^s | x^1, \cdots, x^N) \| r^i(z^s | x^i) \right]$$

$$- \sum_{i=1}^{N} \mathbb{E}_{p_D(x^i)} D_{KL} \left[q(z^i | x^i) \| p(z^i) \right] + \sum_{i=1}^{N} H(X^i). \quad (10)$$

Clearly, there are many terms also present in the ELBO (Eq. 3). We aim to learn a generative model by maximizing the log-likelihood of the joint multilingual distribution while enforcing disagreement by maximizing the regularization in Eq. 10. Thus, we combine these
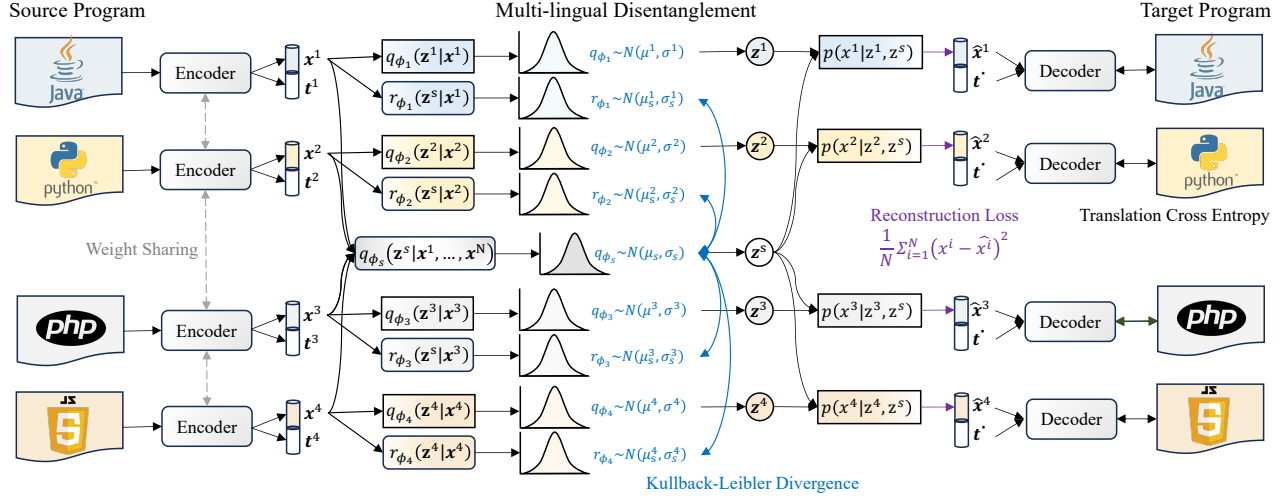
Figure 4: The overall framework.

objectives using a trade-off weight λ:

$$\mathbb{E}_{q(z^1,\cdots,z^N,z^s,x^1,\cdots,x^N)}\left[\log\frac{p(x^1,\cdots,x^N,z^1,\cdots,z^N,z^s)}{q(z^1,\cdots,z^N,z^s|x^1,\cdots,x^N)}\right]$$

$$+ \lambda\sum_{i=1}^{N}\left[I(X^i;\overline{X^i};Z^s)-I(Z^i;Z^s)\right]$$

$$\geq (1+\lambda)\mathbb{E}_{p_D(x^1,\cdots,x^N)}\sum_{i=1}^{N}\mathbb{E}_{q(z^i|x^i)q(z^s|x^1,\cdots,x^N)}[\log p(x^i|z^i,z^s)]$$

$$- (1+\lambda)\mathbb{E}_{p_D(x^i)}\sum_{i=1}^{N}D_{KL}\left[q(z^i|x^i)||p(z^i)\right]$$

$$- \mathbb{E}_{p_D(x^i)}D_{KL}\left[q(z^s|x^1,\cdots,x^N)||p(z^s)\right]$$

$$- \lambda\cdot\mathbb{E}_{p_\mathcal{D}(x^1,\cdots,x^N)}\left[\sum_{i=1}^{N}D_{KL}[q(z^s|x^1,\cdots,x^N)||r^i(z^s_i|x^i)]\right]. \quad (11)$$

The right-hand side (RHS) of Eq. 11 represents the objective for multilingual program translation within a disentangled generative framework.

3.3 The overall framework of VIM-PT

The VIM-PT is architecture-free, which can theoretically and practically be adapted to arbitrary sequence-to-sequence architecture, which involves an encoder and a decoder. Based on the experimental performance, we finally set up a unified encoder for all the languages with weight-sharing and an independent decoder for each programming language. As illustrated in Figure 4, given a multi-parallel sample $(t^1,\cdots,t^N)$ from the dataset $p_\mathcal{D}(t^1,\cdots,t^N)$, where each sample including a group of codes of the same function implemented by the N different programming languages. The flag tokens are initialized by the unused tokens in the tokenizer. To refer to the programming language, we concatenate the flag tokens,

and source code as the input sequence as follows:

$$[[CLS],x_1^n,x_2^n,\cdots,x_k^n,t_1^n,t_2^n,\cdots,t_c^n,[SEP]],n\in[1,N], \quad (12)$$

where k is the length of the flag token, and c is the length of the source code. Then the sequence of the instance is encoded by the weight-sharing encoders to the initial representations. Then the flag representations are inputted into the variational interaction.

After variational interaction, we can reconstruct the flag representations of the other programming languages for each source code of the sample. Then the target flag representations are concatenated with the source code representations and inputted into the target decoder to generate the target code. The regularized training approach learns the model from the objective which consists of the supervised loss function as well as the regularization term described in Eq. 11.

3.4 Learning from multi-parallel samples

The disentangle module involves three approximate posteriors q_{ϕ_i}, r_{ϕ_i}, and p_{ϕ_i}, and a language-shared approximate posterior q_{ϕ_s}. Each approximate posterior is parameterized by a standard Gaussian prior $z\sim\mathcal{N}(0,I)$.

The flag representations of instances can be disentangled by the corresponding language-specific projectors $(q_{\phi_1},\cdots,q_{\phi_N})$ and language-shared projectors q_{ϕ_s}, respectively. In addition, the shift-shared representations are inferred by corresponding shift-shared projectors $(r_{\phi_1},\cdots,r_{\phi_N})$. Then the language-specific representations and the language-shared representation are inputted into $(p_{\phi_1},\cdots,p_{\phi_N})$ to reconstruct the representations of the multilingual instances.

The language-specific representations, the language-shared representation, and the shift-shared representations are denoted by $(\mathbf{z}^1,\cdots,\mathbf{z}^N)$, $\mathbf{z}^s$, and $(\mathbf{z}^s_1,\cdots,\mathbf{z}^s_N)$, respectively. Instructed by Equation 11, the objective of the training of multi-parallel samples is formalized as:

$$\mathcal{L} = ((1+\lambda) \cdot [-\sum_{i=1}^{N} \mathbb{E}_{q(\mathbf{z}^i|\mathbf{x}^i)q(\mathbf{z}^s|\mathbf{x}^1,\cdots,\mathbf{x}^N)} [\underbrace{\log p(\mathbf{x}^i|\mathbf{z}^i,\mathbf{z}^s)}_{\text{Reconstruction Loss}}$$

$$+ \underbrace{\log p_D(\mathbf{t}^i|\hat{\mathbf{x}}^i,\mathbf{t}^s)]]) \quad (13)}_{\text{Translation Cross Entropy}}$$

$$+ (1+\lambda) \cdot \sum_{i=1}^{N} D_{KL}[q(\mathbf{z}^i|\mathbf{x}^i)||p(\mathbf{z}^i)]$$

$$+ D_{KL}[q(\mathbf{z}^s|\mathbf{x}^1,\cdots,\mathbf{x}^N)||p(\mathbf{z}^s)]$$

$$+ \lambda \cdot \sum_{i=1}^{N} D_{KL}[q(\mathbf{z}^s|\mathbf{x}^1,\cdots,\mathbf{x}^N)||r^i(\mathbf{z}_i^s|\mathbf{x}^i)],$$

The first term of the Eq. 13 includes two parts, where the reconstruction loss (Mean Square Error) and translation cross-entropy loss are to reconstruct the representations of flag tokens and source code tokens, respectively. The last term is to minimize the KL divergence between the shift-shared and language-shared representations, so the language-shared representation can be replaced approximately by the shift-shared representation when some instances are missing in the partially missing samples.

3.5 Learning from partially missing samples

For partially missing samples, we design a training strategy to exploit the data that is not multi-parallel. The training strategy is described in Algorithm 1.

At first, given a partially missing sample from the dataset, there may be some missing instances in one sample. To obtain the pseudo instances, we disentangle the language-specific representations and the language-shared representations to generate the draft translation of the missing instances. We sample a set of pseudo instances $(\tilde{t}^1,\cdots,\tilde{t}^N)$ from the training dataset randomly to fill the missing instances of the corresponding language in the original sample. And the flag representations can be separated as $(\tilde{\mathbf{x}}^1,\cdots,\tilde{\mathbf{x}}^N)$. Similar to the learning of multi-parallel samples, the language-specific representations, and the shift-shared representations are disentangled as $(\tilde{\mathbf{z}}^1,\cdots,\tilde{\mathbf{z}}^N)$, and $(\tilde{\mathbf{z}}_1^s,\cdots,\tilde{\mathbf{z}}_N^s)$. However, as some instances of the sample are missing, the language-shared representation can not be inferred.

To overcome this issue, the method generates the draft translation of the missing instances with the pseudo instances. Each language-specific representation of the pseudo instance and the language-shared representation of ground truth are inputted into $(p_{\phi_1},\cdots,p_{\phi_N})$ to reconstruct the representations of the multilingual instances. As the language-shared representation of ground truth is not unique, for each language-shared representation of $(\tilde{\mathbf{z}}_1^s,\cdots,\tilde{\mathbf{z}}_N^s)$, a multi-parallel sample $(\hat{\mathbf{x}}^1,\cdots,\hat{\mathbf{x}}^N$ reconstructed by language-shared information and language-specific information can be obtained. Then the learning of multi-parallel samples can be performed. Especially, only the target programs with ground truth are reconstructed.

Algorithm 1: Pseudocode of training of partially missing samples.

Input: Dataset $\mathcal{D}$ from N programming languages, opt is the optimizer
Output: Model(θ,ϕ)

1 **for** $epoch < max\_epoch$ **do**
2   Draw a partially missing sample from the dataset
3   Construct the pseudo sample $(\tilde{t}^1,\cdots,\tilde{t}^N)$ by filling the missing instance from training set randomly
4   Encode the sample by encoders as $(\tilde{\mathbf{x}}^1,\cdots,\tilde{\mathbf{x}}^N)$
5   Enforce disentanglement on pseudo instances to obtain language-specific representation $(\tilde{\mathbf{z}}^1,\cdots,\tilde{\mathbf{z}}^N)$ and shift-shared representation $(\tilde{\mathbf{z}}_1^s,\cdots,\tilde{\mathbf{z}}_N^s)$ by $(q_{\phi_1},\cdots,q_{\phi_N})$ and $(r_{\phi_1},\cdots,r_{\phi_N})$, respectively
6   Compute $\sum_{i=1}^{N} D_{KL}[q_{\phi_i}(\tilde{\mathbf{z}}^i|\tilde{\mathbf{x}}^i)||p(\tilde{\mathbf{z}}^i)]$
7   **for** *each* $\tilde{\mathbf{z}}_i^s$ *obtained by* r_{ϕ_i} **do**
8   Reconstruct the sample $(\mathbf{x}^1,\cdots,\mathbf{x}^N)$ with $(\tilde{\mathbf{z}}^1,\cdots,\tilde{\mathbf{z}}^N)$ and $\tilde{\mathbf{z}}_i^s$ by $(p_{\phi_1},\cdots,p_{\phi_N})$
9   Enforce disentanglement on multi-parallel instances to obtain language-specific representation $(\mathbf{z}^1,\cdots,\mathbf{z}^N)$ and language-shared representation $\mathbf{z}^s$ by $(q_{\phi_1},\cdots,q_{\phi_N})$ and q_{ϕ_s} respectively
10   Compute $\sum_{i=1}^{N} D_{KL}[q_{\phi_i}(\mathbf{z}^i|\mathbf{x}^i)||p(\mathbf{z}^i)]$, $D_{KL}[q_{\phi_s}(\mathbf{z}^s|\mathbf{x}^1,\cdots,\mathbf{x}^N)||p(\mathbf{z}^s)]$ and $\sum_{i=1}^{N} D_{KL}[q_{\phi_s}(\mathbf{z}^s|\mathbf{x}^1,\cdots,\mathbf{x}^N)||r_{\phi_i}(\mathbf{z}^s|\mathbf{x}^i)]$
11   Reconstruct the instance representation $(\hat{\mathbf{x}}^1,\cdots,\hat{\mathbf{x}}^N)$ with $(\mathbf{z}^1,\cdots,\mathbf{z}^N)$ and $\mathbf{z}^s$ by $(p_{\phi_1},\cdots,p_{\phi_N})$
12   Generate the target programs with ground truth by decoders and compute the first term of Eq. 13.
13   **end**
14      $\theta,\phi \leftarrow opt(\theta,\phi,\nabla_{\theta,\phi})$
15 **end**

---

4 EXPERIMENTS

4.1 Dataset Description

We conduct experiments on the *CoST* dataset [51], which is a large and comprehensive dataset to evaluate the performance of program translation approaches. The dataset consists of both snippet-level and program-level parallel data from 7 programming languages (i.e., C, C#, C++, Java, JavaScript, PHP, and Python) and up to 42 programming language pairs, which was collected from the GeeksForGeeks website [1]. The platform ensures its contributors stick to a template in terms of the comments used in their programs and the code corresponding to those comments. The dataset provides a good number of multilingual instances of code that can be effectively used for this task. The detailed statistics of the dataset are shown in Table 1, while the train, validation, and test sets are split the same as *CoST* [51]. For each language, there are some instances of corresponding samples that are missing, especially in C and PHP.

4.2 Experimental Settings

We choose the following state-of-the-art multi-lingual and pairwise program translation methods as baselines. All the encoder-only pretrained models are fine-tuned on the task with the same decoder as our methods.

Table 1: Sample counts for each language in CoST. Py denotes Python, and JS denotes JavaScript.

Granularity	Dataset	Java	C#	C++	C	Py	PHP	JS	Samples
Program	train	1,442	1,382	1,442	183	1,343	435	904	1,507
	val	49	49	49	49	49	49	49	49
	test	69	69	69	69	69	69	69	69
Snippet	train	13,979	47,722	27,101	4,517	34,582	12,752	28,376	49,962
	val	272	1,524	544	813	948	1,402	1,185	1,524
	test	411	2,301	823	1,238	1,423	2,138	1,778	2,308
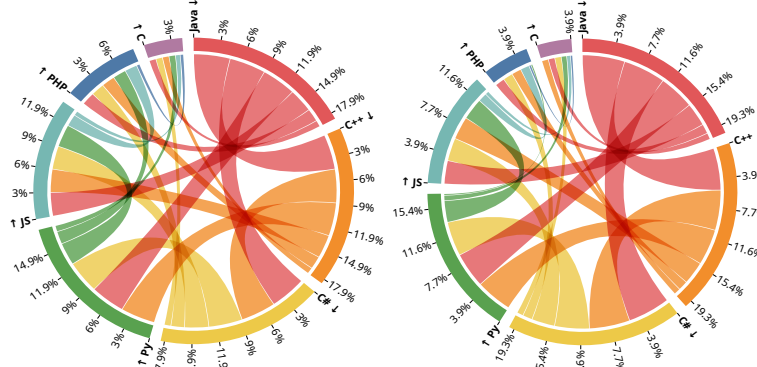
Figure 5: Program Pairs on CoST.   Figure 6: Snippet Pairs on CoST.

Table 2: Pairwise statistics of the CoST. The upper triangle (in normal font) shows the number of parallel snippets, while the lower triangle (in bold font) shows the number of parallel programs. JS represents JavaScript and Py represents Python.

Lang	C	C#	C++	Java	JS	PHP	Py
C	-	2123	2188	2135	1232	700	1779
C#	**273**	-	13326	13905	7601	3192	11404
C++	**267**	**1442**	-	13929	7596	3165	11930
Java	**281**	**1495**	**1497**	-	7729	3194	11713
JS	**196**	**994**	**996**	**1009**	-	2917	7165
PHP	**135**	**552**	**548**	**552**	**512**	-	545
Py	**263**	**1383**	**1419**	**1417**	**962**	**545**	-

- **Naïve Copy**: A direct copy from the source program to the translation output, which denotes how similar the source language and the target language are.
- **DOBF** [19]: A pre-training model for programming languages, which leverages the structural aspect of programming languages and recovers the original version of obfuscated source code.
- **CodeBERT** [12]: A bimodal pre-training model, trained with a hybrid objective function that incorporates the pre-training task of standard masked language modeling and replaced token detection in the pre-training stage.
- **MuST-PT** [51]: The state-of-the-art multilingual program translation technique that utilizes multilingual languages and exhibits strong generalizability enhances translation performance, particularly for low-resource languages.
- **M-PT(*w/o* VI)**: A straightforward approach of multilingual program translation. where each programming language shares an encoder and has an independent decoder. This approach is a blank controller of VIM-PT without variational interaction.

To compare with these baselines, we follow the best hyper-parameters suggested in their studies. For hyper-parameters in our method, the numbers of transformer layers of the encoder and decoder are set as 12 and 6, respectively. The model dimension and attention heads in transformer layers are set as 768 and 12. The hyper-parameters α, k are determined based on the performance of the validation set, which is set as 1e-3, 16 in the experiment. Then, the training set and the validation set are mixed up to train the model. Following MuST-PT[51], the snippet-level training set is utilized to enhance the program-level translation, and some data of C++ is used to supplement C. Following the baselines [35, 51],

the comments in the source code are kept in the dataset, which can increase the number of anchor points across languages, and the parameters of CodeBERT [12] are utilized to initialize the encoder to accelerate the training process. The AdamW [24] optimizer is used to update model parameters with the initial learning rate 1e-4. The linear weight decay is used for scheduling the learning rate. The process is repeated for 3 times and the average performance on the test set is reported. All the experiments are conducted with the NVIDIA Tesla A100 with 128GB RAM on the Ubuntu system.

4.3 RQ1: What is the Performance of VIM-PT Comparing With Baseline Approaches?

In the experiment, we use the BLEU [33] score as the evaluation metric to evaluate the n-gram overlap between the translated code and the ground-truth target code, which is the most widely used metric in program translation [13, 19, 25, 51]. A higher BLEU score indicates better evaluation performance, which varies from 0 to 100 as a percentage. Table 3 shows the experimental results. For each pair of translations, the three best performing approaches are highlighted, while dark gray marks an approach with the highest retrieval performance, and gray and light gray correspond to the second- and third-best approaches. Compared with the pairwise approach like CodeBERT with a similar backbone, the multilingual approaches including MuST-PT, VIM-PT(w/o VT), and VIM-PT perform significantly outperformance. We can attribute this result to (1) expansion of the representation scope, as the different language corpora face various application scenarios and have different functional preferences, (2) better latent space, which can be jointly constructed

## Table 3: Performance evaluation in terms of BLEU-4.

Lang	Method	Snippet Level							Program Level						
		C	C#	C++	Java	JS	PHP	Py	C	C#	C++	Java	JS	PHP	Py
C	Naïve Copy	-	68.88	85.58	69.17	54.85	37.48	37.84	-	68.74	85.02	69.34	53.90	38.1	36.09
	DOBF	-	39.38	40.17	41.57	33.54	34.51	17.93	-	27.64	20.90	27.15	20.77	24.15	25.87
	CodeBERT	-	51.92	60.84	51.70	40.57	34.49	31.49	-	34.41	33.64	33.75	29.90	29.32	27.34
	MuST-PT	-	80.68	88.58	79.24	80.35	82.94	66.49	-	78.39	84.92	76.84	66.13	70.62	55.71
	M-PT(w/o VI)	-	76.82	84.65	70.57	73.86	72.83	65.94	-	74.76	76.22	72.28	70.91	69.57	59.62
	VIM-PT	-	81.21	89.80	79.85	80.98	85.86	69.47	-	81.19	85.64	78.83	73.81	73.82	65.48
C#	Naïve Copy	68.91	-	67.29	78.13	58.90	35.01	36.49	68.74	-	67.51	78.69	57.61	35.55	34.62
	DOBF	38.33	-	38.94	47.84	28.70	49.32	25.14	26.38	-	27.50	31.63	23.62	34.90	22.94
	CodeBERT	52.65	-	79.28	83.90	76.99	6	64.72	34.93	-	65.74	80.11	53.72	45.67	47.14
	MuST-PT	81.12	-	85.34	85.80	82.74	81.64	71.11	78.78	-	84.72	87.76	70.00	70.66	62.03
	M-PT(w/o VI)	79.41	-	81.24	83.75	78.64	75.83	72.16	69.53	-	74.87	80.80	71.63	68.38	61.09
	VIM-PT	82.02	-	85.92	83.99	83.15	85.27	73.42	79.36	-	84.82	87.63	81.41	74.26	65.78
C++	Naïve Copy	85.66	67.33	-	67.32	55.44	37.68	36.92	85.02	67.51	-	67.38	54.07	38.47	34.89
	DOBF	43.32	42.25	-	42.03	40.01	49.29	25.77	31.84	37.43	-	48.70	34.05	15.67	23.73
	CodeBERT	63.24	77.21	-	78.39	75.10	70.75	68.92	45.57	64.15	-	56.47	48.65	40.59	56.73
	MuST-PT	87.55	82.98	-	80.27	81.01	83.29	71.20	84.20	81.15	-	79.15	68.85	71.18	64.10
	M-PT(w/o VI)	83.40	83.54	-	81.67	78.91	77.71	69.92	71.75	74.43	-	72.57	71.19	69.94	61.27
	VIM-PT	86.57	86.19	-	84.69	82.35	83.56	73.21	77.11	83.27	-	79.92	75.63	73.89	66.07
Java	Naïve Copy	69.14	78.03	67.25	-	57.33	33.82	35.50	69.40	78.77	67.48	-	55.99	33.66	33.60
	DOBF	39.21	44.26	38.80	-	40.23	48.87	24.83	32.23	65.02	22.01	-	55.78	35.75	24.90
	CodeBERT	54.98	86.02	79.14	-	78.54	70.21	66.41	46.85	80.88	69.88	-	55.15	47.66	48.56
	MuST-PT	81.16	90.13	85.23	-	81.87	80.39	70.06	78.71	89.93	84.28	-	69.53	69.83	61.12
	M-PT(w/o VI)	78.54	90.36	81.07	-	80.55	72.31	74.03	70.25	78.66	75.54	-	70.80	69.20	61.59
	VIM-PT	81.96	93.12	85.79	-	83.29	80.25	78.29	82.09	90.43	84.79	-	75.89	74.25	66.04
JS	Naïve Copy	54.58	58.61	55.29	56.61	-	30.44	41.58	53.00	56.70	53.29	54.44	-	31.53	39.77
	DOBF	33.16	41.50	39.91	44.16	-	46.93	24.05	22.13	38.06	20.69	37.78	-	26.03	21.21
	CodeBERT	40.93	75.38	73.83	74.58	-	63.85	60.99	32.88	58.43	50.42	60.13	-	46.14	44.34
	MuST-PT	78.54	78.91	78.95	78.03	-	78.69	66.47	70.20	73.32	73.01	73.39	-	76.44	63.88
	M-PT(w/o VI)	75.90	80.44	77.40	79.23	-	78.56	66.53	65.99	71.93	72.57	70.08	-	64.68	56.61
	VIM-PT	79.96	81.76	78.09	79.27	-	79.51	67.02	70.64	79.07	75.05	74.07	-	77.57	64.72
PHP	Naïve Copy	37.46	34.99	37.64	33.85	30.66	-	23.67	38.10	35.55	38.47	33.61	32.01	-	23.04
	DOBF	25.78	40.88	38.30	42.98	38.11	-	25.64	17.62	31.95	30.18	25.88	25.89	-	20.80
	CodeBERT	30.06	65.67	67.68	64.02	62.06	-	57.01	30.82	47.14	43.04	45.83	43.45	-	39.42
	MuST-PT	76.67	77.96	79.41	76.42	77.64	-	69.34	67.88	70.34	70.04	67.30	73.54	-	63.97
	M-PT(w/o VI)	73.68	75.04	76.02	68.10	68.20	-	61.25	68.33	72.24	73.42	70.49	69.12	-	61.08
	VIM-PT	78.30	78.79	81.37	77.02	76.76	-	69.65	70.58	73.82	80.30	73.64	76.76	-	70.44
Py	Naïve Copy	37.77	36.42	36.90	35.24	41.53	23.59	-	35.74	34.31	34.62	33.00	39.79	22.85	-
	DOBF	28.77	34.07	36.23	33.48	30.71	45.68	-	15.47	29.22	24.99	35.64	27.31	28.21	-
	CodeBERT	35.82	61.50	71.06	65.99	62.34	63.73	-	53.17	49.16	57.63	52.93	52.30	45.69	-
	MuST-PT	70.64	72.35	75.37	70.89	70.46	75.49	-	58.70	63.23	66.16	64.57	66.47	70.90	-
	M-PT(w/o VI)	63.72	67.88	72.39	67.54	69.43	66.78	-	63.72	67.88	68.39	67.54	59.43	66.78	-
	VIM-PT	71.11	77.68	75.85	73.45	72.74	76.05	-	64.65	68.53	69.59	67.64	68.29	71.00	-

by different languages to represent the language-sharing semantics, and (3) the auxiliary benefit to low-resource language with rich-resource language, which is last but most important factor. In practical scenarios, where different language corpora are unevenly distributed, building a low-resource programming language to latent space mapping can be benefited from learning among other rich-resource programming languages, which is more effective and robust than building a pairwise mapping using only parallel data.

It can be observed that VIM-PT performs performance gains widely over the state-of-the-art approaches on most of the translation pairs, especially at the program level. VIM-PT performs better with the improvement of 4.57 and 1.87% over the state-of-the-art approach in multi-lingual program translation [51] at the program level and snippet level, respectively. In particular, VIM-PT performs better than all the pairwise approaches and multi-lingual approaches on all the translations when Python is the source language or target language. Table 4 further calculates the average

Table 4: The average BLEU-4 score of the translation of different approaches *from* or *to* the languages.

Method	Snippet Level								Program Level							
from	C	C#	C++	Java	JS	PHP	Py	Avg	C	C#	C++	Java	JS	PHP	Py	Avg
Naïve	58.97	57.46	58.39	56.85	49.52	33.05	35.24	49.93	58.53	57.12	57.89	56.48	48.12	33.46	33.39	49.28
DOBF	34.52	38.05	40.45	39.37	38.29	35.28	34.82	37.25	24.41	27.83	31.90	39.28	27.65	25.39	26.81	29.04
CodeBERT	45.17	71.03	72.27	72.55	64.93	57.75	60.07	63.40	31.39	54.55	52.03	58.16	48.72	41.62	51.81	48.33
MuST-PT	79.71	81.29	81.05	81.47	76.60	76.24	72.53	78.41	72.10	75.66	74.77	75.57	71.71	68.85	65.01	71.95
M-PT(*w/o* VI)	74.11	78.50	79.19	79.48	76.34	70.38	67.96	75.14	70.56	71.05	70.19	71.01	66.98	69.11	65.62	69.22
VIM-PT	81.12	82.30	82.76	83.78	77.60	76.98	74.48	79.86	76.46	78.88	75.98	78.92	73.52	74.26	68.28	75.19
Improvement(%)	9.46	4.84	4.51	5.41	1.65	9.38	9.59	6.41	8.36	11.02	8.25	11.14	9.76	7.45	4.05	8.58
to	C	C#	C++	Java	JS	PHP	Py	Avg	C	C#	C++	Java	JS	PHP	Py	Avg
Naïve	58.92	57.38	58.33	56.72	49.79	33.00	35.33	49.92	58.33	56.93	57.73	56.08	48.90	33.36	33.67	49.29
DOBF	34.76	40.39	38.73	42.01	35.22	45.77	23.89	37.25	24.28	38.22	24.38	34.46	31.24	27.45	23.24	29.04
CodeBERT	46.28	69.62	71.97	69.76	65.93	61.94	58.26	63.39	40.70	55.70	53.39	54.87	47.20	42.51	43.92	48.33
MuST-PT	79.28	80.50	82.15	78.44	79.01	80.41	69.11	78.41	73.08	76.06	77.19	74.84	69.09	71.61	61.80	71.95
M-PT(*w/o* VI)	75.78	79.01	78.80	75.14	74.93	74.00	68.31	75.14	68.26	73.32	73.50	72.29	68.85	68.09	60.21	69.22
VIM-PT	79.99	83.13	82.80	79.71	79.88	81.75	71.84	79.87	74.07	79.39	80.03	76.96	75.30	74.13	66.42	75.19
Improvement(%)	5.56	5.20	5.09	6.08	6.60	10.47	5.18	6.31	8.51	8.28	8.88	6.45	9.37	8.87	10.32	8.67
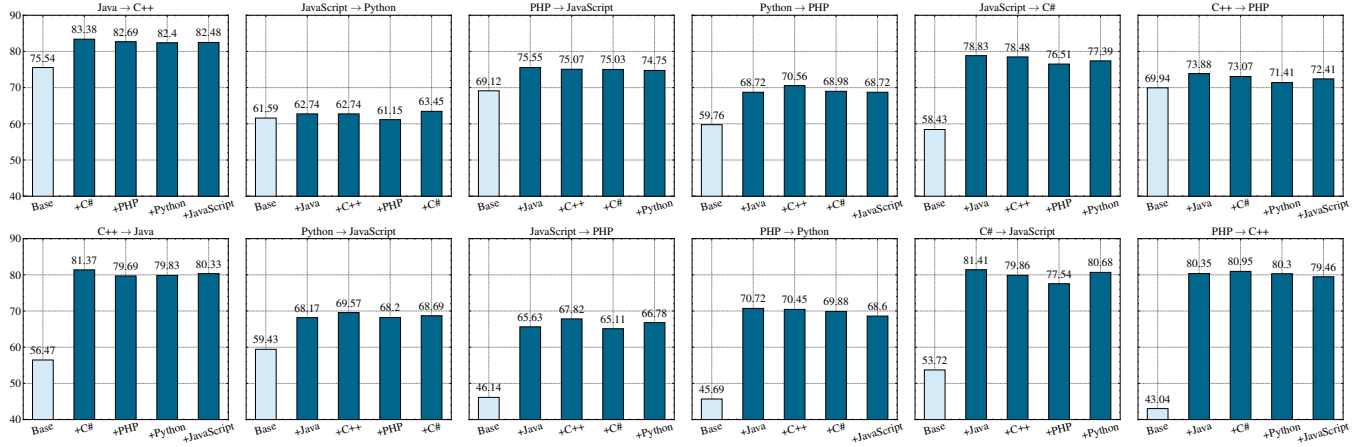
Figure 7: The auxiliary benefit of different languages in program translation.

score of the translation of different approaches *from* or *to* any programming languages based on Table 3. As shown in Table 4, VIM-PT has performed the state-of-the-art at both snippet and program level in all configurations of the average score *from* or *to* any programming languages. We notice the performance of MusT-PT is lower than M-PT(*w/o* VI) on PHP and Python, which indicates that using the weight-sharing decoder is not conducive to the modeling of low-resource languages with few samples, which may be overwhelmed by the rich-resource languages in the joint training.

4.4 RQ2: What is the impact of Variational Interaction for the VIM-PT?

To evaluate the effectiveness of the variational interaction, we set a blank controller termed M-PT(*w/o* VI), which ablates variational interaction on VIM-PT. As shown in Table 3 and Table 4, the inclusion of Variational Interaction (VI) in VIM-PT results in an improvement

in the average BLEU-4 score at both snippet Level and program Level. Across most programming language pairs, VIM-PT achieves higher performance than M-PT(*w/o* VI), indicating that variational interaction indeed enhances translation quality.

In summary, the incorporation of VI effectively enhances the performance of the translation model, resulting in more accurate and fluent translations. The *Improvement* in Table 4 indicate the average improvement of the variational interaction across all languages and levels, with an average increase from 4.05% to 11.14%, and from 1.65% to 10.47% at the program level and snippet level, respectively. The variational interaction leads to notable improvements especially in low-resource languages like C and PHP, indicating that variational interaction plays a crucial role in enhancing the performance of the low-resource languages in multilingual translation.

4.5 RQ3: Which rich-resource languages benefit the low-resource languages?

Due to the diversity between different programming languages, we explore the auxiliary benefits of different programming languages as rich-resource to the translation between low-resource programming languages. As shown in Figure 7, with auxiliary of any other programming languages, the performance of each direction of program translation has obtained a significant improvement. It is because when importing the third programming language, more data are utilized in the learning and the language-shared latent space can be constructed better. The base indicates the pair-wised approach based on CodeBERT and the others are the translation with the help of the other languages based on the framework of VIM-PT. The auxiliary benefits of different programming languages may depend on the following possible factors.

4.5.1 The imbalance of the data of different auxiliary languages. The amount of data of auxiliary language has a significant influence on the auxiliary benefit of program translation. As mentioned above, more data are utilized in the learning and the language-shared latent space can be constructed better, and further influence the decoding of the language-shared features. As illustrated in Figure 5, compared to PHP the number of pairs from Java, C++, and C# is larger. Therefore, when Java, C++, and C# are selected as auxiliary languages, the benefits for the translation are more significant than PHP as the auxiliary language generally.

4.5.2 Compiled and interpreted programming languages. Programming languages can be divided into compiled and interpreted types, depending on how programs are executed. Compiled languages require that all source code be translated into machine code when writing a program, which is then executed by a computer. Common compiled languages include C, C++, and Java. Interpreted languages translate source code into machine code line by line or block by block when executing a program and execute it. Common interpreted languages include Python, JavaScript, and PHP. As shown in Figure 7, in the compiled-compiled translation from C++ to Java, using the compiled language C# as auxiliary can achieve higher performance than the other interpreted languages, such as PHP, Javascript, and Python.

4.5.3 Syntax Similarity. Although each programming language has a unique formal syntax, the syntax similarity between different languages is various. For instance, C# is similar to C++ in syntax, while quite different from Python. As shown in Figure 7, in the translation from PHP to C++, with the help of C#, the improvement of the performance is more significant than Python as an auxiliary.

In summary, it is observed that any language that serves as an auxiliary generally has a supporting role, which demonstrates the effectiveness of VIM-PT in addressing the distribution shift of semantics across multiple languages. Yet many possible factors influence the benefit of the auxiliary language, thus how to select the rich-resource languages as auxiliary languages is still a question worth exploring in the multi-lingual program translation.
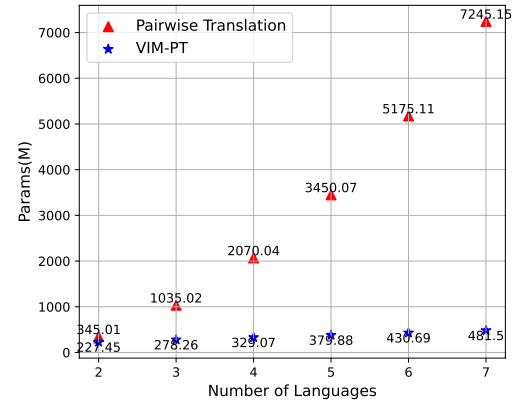

Figure 8: The comparison of space overhead between pairwise approach and VIM-PT in terms of parameters.

4.6 RQ4: What is the deployment complexity of the joint model and pairwise approaches?

To compare the deployment complexity between the pairwise approach and VIM-PT, we record the space overhead of the translation model. As shown in Figure 8, the pairwise approach and VIM-PT are both constructed with CodeBERT as encoder and the 6-layer Transformer as Decoder. With the number of languages in translation increasing, the number of parameters of the pairwise approach grows exponentially, and VIM-PT grows linearly.

It is because the encoder-decoder models needed by the pairwise approach are coupling with the number of translation directions, but the decoders needed by VIM-PT are coupling with the number of languages. In the bilingual program translation, the pairwise approach should construct two models for the two directions in translation, and the number of parameters of the pairwise approach and VIM-PT is similar. When the multilingual translation gradually increases to 7 languages, the pairwise approach should construct 42 encoder-decoder models for the 42 directions in translation. In this case, VIM-PT only needs one weight-sharing encoder and 7 decoders, where the number of parameters is less than 1/15 of the pairwise approaches. It demonstrates the space efficiency of multilingual program translation.

5 THREAT ANALYSIS

Our results are interpreted with two threats to validity in mind.

- The internal threat to validity lies in the implementation of compared techniques. To reduce it, we directly reuse the implementation of the compared techniques from their reproducible packages and the weights of pre-trained models, if they are available and executable. Otherwise, we reimplement the techniques strictly following the papers on existing mature libraries.
- The external threat to validity lies in the dataset used in the experiment. To mitigate the external threat, the widely-used dataset is used to evaluate the effectiveness of the method, which includes 7 programming languages and up to 42 pairs and two levels. The artifact contains the dataset and the code is publicly available in the supplementary material.

6 CONCLUSION

In this paper, we argue that jointly learning a unified model to translate code across multiple programming languages is superior to separately learning from bilingual parallel data. We propose Variational Interaction for Multilingual Program Translation (VIM-PT), a disentanglement-based generative approach that jointly trains a unified model for multilingual program translation across multiple languages. VIM-PT disentangles code into language-shared and language-specific features using variational inference and interaction information with a novel lower bound. With the variational interaction, VIM-PT achieves significant improvement in multilingual program translation, mines and leverages the capability of non-parallel data, addresses the distribution shift of program semantics across languages, and serves as a unified model, reducing deployment complexity. In the future, more effective selection strategies in filling the partially missing samples can be explored beyond the random selection used in our method.

7 ACKNOWLEDGMENTS

A APPENDICES

A.1 $I(Z^i; Z^s) = -I(X^i; Z^i, Z^s) + I(X^i; Z^i) + I(X^i; Z^s)$

With the interaction information between three random variables:

$$I(X; Y; Z) = I(X; Z) - I(X; Z|Y) = I(Y; Z) - I(Y; Z|X), \quad (14)$$

we can obtain the mutual information between Z^i and Z^s:

$$I(Z^i, Z^s) = I(Z^i; X^i) - I(Z^i; X^i|Z^s) + I(Z^i; Z^s|X^i). \quad (15)$$

Due to the structural assumption on q, the $q(z^i|x^i) = q(z^i|x^i, z^s)$ holds, so the last term in the above function is eliminated:

$$
\begin{aligned}
I(Z^i; Z^s|X^i) &= H(Z^i|X^i) - H(Z^i|X^i, Z^s) \\
&= H(Z^i|X^i) - H(Z^i|X^i) = 0,
\end{aligned} \quad (16)
$$

which yields

$$
\begin{aligned}
I(Z^i, Z^s) &= I(X^i; Z^i) - I(X^i; Z^i|Z^s) \\
&= I(X^i; Z^i) + I(X^i; Z^s) - I(X^i; Z^i, Z^s).
\end{aligned} \quad (17)
$$

REFERENCES

[1] [n. d.]. https://www.geeksforgeeks.org/.
[2] Wasi Uddin Ahmad, Md Golam Rahman Tushar, Saikat Chakraborty, and Kai-Wei Chang. 2023. AVATAR: A Parallel Corpus for Java-Python Program Translation. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023.* Association for Computational Linguistics, 2268–2281.
[3] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. 2017. Deep Variational Information Bottleneck. In *5th International Conference on Learning Representations, Toulon, France, April 24-26.*
[4] Miltiadis Allamanis, Earl T. Barr, Premkumar T. Devanbu, and Charles Sutton. 2018. A Survey of Machine Learning for Big Code and Naturalness. *ACM Comput. Surv.* (2018), 81:1–81:37.
[5] Khalid Alsubhi, Fawaz Alsolami, A Algarni, E Albassam, Maher Khemakhem, F Eassa, K Jambi, and M Usman Ashraf. 2019. A Tool for Translating sequential source code to parallel code written in C++ and OpenACC. In *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA).* IEEE, 1–8.
[6] Kijin An, Na Meng, and Eli Tilevich. 2018. Automatic inference of java-to-swift translation rules for porting mobile applications. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems.* 180–190.
[7] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree Neural Networks for Program Translation. In *Advances in Neural Information Processing Systems 31, Montréal, Canada.* 2552–2562.
[8] William C. Chu. 1993. A Re-Engineering Approach to Program Translation. In *Proceedings of the Conference on Software Maintenance, ICSM 1993, Montréal, Quebec, Canada.* IEEE Computer Society, 42–50.
[9] Yali Du, Yi-Fan Ma, Zheng Xie, and Ming Li. 2023. Beyond Lexical Consistency: Preserving Semantic Consistency for Program Translation. In *IEEE International Conference on Data Mining, ICDM 2023, Shanghai, China, December 1-4, 2023.* 91–100.
[10] Yali Du and Zhongxing Yu. 2023. Pre-training Code Representation with Semantic Flow Graph for Effective Bug Localization. *CoRR* abs/2308.12773 (2023).
[11] Bryan Eikema and Wilker Aziz. 2018. Auto-encoding variational neural machine translation. *arXiv preprint arXiv:1807.10564* (2018).
[12] Zhangyin Feng, Daya Guo, Du-Yu Tang, Nan Duan, Xiao-Cheng Feng, Ming Gong, Lin-Jun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP, Online Event.* Association for Computational Linguistics, 1536–1547.
[13] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *9th International Conference on Learning Representations, Virtual Event, Austria.*
[14] HyeongJoo Hwang, Geon-Hyeong Kim, Seunghoon Hong, and Kee-Eung Kim. 2020. Variational interaction information maximization for cross-domain disentanglement. *Advances in Neural Information Processing Systems* 33 (2020), 22479–22491.
[15] Svetoslav Karaivanov, Veselin Raychev, and Martin T. Vechev. 2014. Phrase-Based Statistical Translation of Programming Languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, part of SPLASH '14, Portland, OR, USA.* ACM, 173–184.
[16] Thomas R Kennedy III. 1987. Using program transformation to improve program translation. (1987).
[17] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. *Advances in neural information processing systems* 27 (2014).
[18] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, Banff, AB, Canada, April 14-16,* Yoshua Bengio and Yann LeCun (Eds.).
[19] Marie-Anne Lachaux, Baptiste Rozière, Marc Szafraniec, and Guillaume Lample. 2021. DOBF: A Deobfuscation Pre-Training Objective for Programming Languages. In *Advances in Neural Information Processing Systems 34, virtual.*
[20] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018. Unsupervised Machine Translation Using Monolingual Corpora Only. In *6th International Conference on Learning Representations, Vancouver, BC, Canada.* OpenReview.net.
[21] Robert S Ledley and James B Wilson. 1962. Automatic-programming-language translation through syntactical analysis. *Commun. ACM* 5, 3 (1962), 145–155.
[22] Zhifei Li, Jason Eisner, and Sanjeev Khudanpur. 2009. Variational decoding for statistical machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP.* 593–601.
[23] Jiaqi Liu, Fengming Zhang, Xin Zhang, Zhiwen Yu, Liang Wang, Yao Zhang, and Bin Guo. 2024. hmCodeTrans: Human-Machine Interactive Code Translation. *IEEE Transactions on Software Engineering* (2024).
[24] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations,.* OpenReview.net.
[25] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, virtual.*
[26] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen. 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'13, Saint Petersburg, Russian Federation.* ACM, 651–654.
[27] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen. 2015. Divide-and-Conquer Approach for Multi-phase Statistical Migration for Source Code (T). In *30th IEEE/ACM International Conference on Automated Software Engineering.* IEEE Computer Society, 585–596.
[28] Trong Duc Nguyen, Anh Tuan Nguyen, and Tien N. Nguyen. 2016. Mapping API elements for code migration with vector representations. In *Proceedings of the 38th International Conference on Software Engineering.* ACM.

[29] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 574–584.

[30] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation (T). In *30th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 574–584.

[31] Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation (T). In *30th IEEE/ACM International Conference on Automated Software Engineering, Lincoln, NE, USA*. IEEE Computer Society.

[32] Artidoro Pagnoni, Kevin Liu, and Shangyan Li. 2018. Conditional variational autoencoder for neural machine translation. *arXiv preprint arXiv:1812.04405* (2018).

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA*. ACL, 311–318.

[34] Sergio Pino, Lori Pollock, and Sunita Chandrasekaran. 2017. Exploring translation of OpenMP to OpenACC 2.5: lessons learned. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 673–682.

[35] Baptiste Rozière, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised Translation of Programming Languages. In *Advances in Neural Information Processing Systems, NeurIPS, virtual*.

[36] Baptiste Rozière, Jie Zhang, François Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. 2022. Leveraging Automated Unit Tests for Unsupervised Code Translation. In *ICLR*.

[37] Hendra Setiawan, Matthias Sperber, Udhyakumar Nallasamy, and Matthias Paulik. 2020. Variational Neural Machine Translation with Normalizing Flows. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 7771–7777.

[38] Xin Sheng, Linli Xu, Junliang Guo, Jingchang Liu, Ruoyu Zhao, and Yinlong Xu. 2020. Introvnmt: An introspective model for variational neural machine translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8830–8837.

[39] Jinsong Su, Shan Wu, Deyi Xiong, Yaojie Lu, Xianpei Han, and Biao Zhang. 2018. Variational recurrent neural machine translation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[40] Weisong Sun, Chunrong Fang, Yuchen Chen, Guanhong Tao, Tingxu Han, and Quanjun Zhang. 2022. Code search based on context-aware code translation. In *Proceedings of the 44th International Conference on Software Engineering*. 388–400.

[41] Marc Szafraniec, Baptiste Rozière, Hugh Leather, François Charton, Patrick Labatut, and Gabriel Synnaeve. 2022. Code Translation with Compiler Representations. *CoRR* abs/2207.03578 (2022).

[42] Mudiarta Tauda, Zahir Zainuddin, and Zulkifli Tahir. 2021. Programming Language Translator For Integration Client Application With Web APIs. In *2021 International Conference on Artificial Intelligence and Mechatronics Systems (AIMS)*. IEEE, 1–4.

[43] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 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, Virtual Event / Punta Cana, Dominican Republic*. Association for Computational Linguistics, 8696–8708.

[44] Richard C. Waters. 1988. Program Translation via Abstraction and Reimplementation. *IEEE Trans. Software Eng.* 14, 8 (1988), 1207–1228.

[45] Justin D Weisz, Michael Muller, Steven I Ross, Fernando Martinez, Stephanie Houde, Mayank Agarwal, Kartik Talamadupula, and John T Richards. 2022. Better together? an evaluation of ai-supported code translation. In *27th International conference on intelligent user interfaces*. 369–391.

[46] Justin D. Weisz, Michael J. Muller, Stephanie Houde, John T. Richards, Steven I. Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. 2021. Perfection Not Required? Human-AI Partnerships in Code Translation. In *IUI '21: 26th International Conference on Intelligent User Interfaces, College Station, TX, USA, April 13-17, 2021*. ACM, 402–412.

[47] Yuanbo Wen, Qi Guo, Qiang Fu, Xiaqing Li, Jianxing Xu, Yanlin Tang, Yongwei Zhao, Xing Hu, Zidong Du, Ling Li, Chao Wang, Xuehai Zhou, and Yunji Chen. 2022. BabelTower: Learning to Auto-parallelized Program Translation. In *International Conference on Machine Learning, Baltimore, Maryland, USA*. PMLR.

[48] Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. 2023. CodeTransOcean: A Comprehensive Multilingual Benchmark for Code Translation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*. Association for Computational Linguistics, 5067–5089.

[49] Zaixiang Zheng, Hao Zhou, Shujian Huang, Lei Li, Xin-Yu Dai, and Jiajun Chen. 2020. Mirror-Generative Neural Machine Translation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

[50] Ming Zhu, Ismini Lourentzou, and Danfeng Yao. 2023. Alignment-Enhancing Parallel Code Generation for Semi-Supervised Code Translation. (2023).

[51] Ming Zhu, Karthik Suresh, and Chandan K. Reddy. 2022. Multilingual Code Snippets Training for Program Translation. In *Thirty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press.