# Bridging Code-Text Representation Gap using Explanation

**Hojae Han**[†] STOVECAT@SNU.AC.KR
**Youngwon Lee**[†] LUDAYA@SNU.AC.KR
**Minsoo Kim**[‡] MINSOO9574@YONSEI.AC.KR
**Seung-won Hwang**[†][*] SEUNGWONH@SNU.AC.KR
[†]*Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Republic of Korea*
[‡]*Yonsei University, 50 Yonsei-ro, Seodaemun-gu, Seoul 03722, Republic of Korea*

**Editors:** Vineeth N Balasubramanian and Ivor Tsang

## Abstract

This paper studies Code-Text Representation (CTR) learning, aiming to learn general-purpose representations that support downstream code/text applications such as code search, finding code matching textual queries. However, state-of-the-arts do not focus on matching the gap between code/text modalities. In this paper, we complement this gap by providing an intermediate representation, and view it as "explanation." Our contribution is three fold: First, we propose four types of explanation utilization methods for CTR, and compare their effectiveness. Second, we show that using explanation as the model input is desirable. Third, we confirm that even automatically generated explanation can lead to a drastic performance gain. To the best of our knowledge, this is the first work to define and categorize code explanation, for enhancing code understanding/representation.

**Keywords:** Code-Text Representation Learning, Code Search, Code Explanation

## 1. Introduction

This paper studies Code-Text Representation (CTR) learning, for semantically grounding code-text pairs, projecting to the same location in the geometric embedding space. This representation would immediately enable cross-modal search: One may encode some code, to find the closest textual description, and alternatively, one may encode some textual description to search the matching code example, e.g., using "Show me the code of bubble sort" to retrieve the function that fits the description. This would be a productivity booster for software development tools such as VisualStudio.

One core problem of CTR learning is the modality gap between natural language descriptions and runnable raw codes. Table 4 shows this gap as the text example and the code example share few tokens, though they are semantically aligned (the code answers the text). However, current CTR learning approaches, such as CodeBERT Feng et al. (2020), GraphCodeBERT Guo et al. (2020) and PLBART Ahmad et al. (2021), do not focus on the mismatch. To end this, we consider "Intermediate Representation" (IR) between the two: Guo et al. (2019) is an example showing that using IR is effective for text-SQL representation. We extend this concept into a more general problem of CTR via using pseudocode

[*] corresponding author, supported by Microsoft Research Asia and IITP grants (2021-0-01696, High-Potential Individuals Global Training Program, and 2021-0-01343, SNU AI Graduate School)

as another form of IR. A pseudocode is a human-readable form of a given code for human consumption. Another characteristic of pseudocode is that it is code-generic, unlike IR in Guo et al. (2019) which is specific to SQL as a target.

To explore a solution space of IR, our distinction is to treat IR as an "explanation," i.e., additional information helping model to predict answers. Following previous approaches of providing explanation to machine models, which have been actively studied to complement supervised learning, we probe the solution space considered for leveraging explanation for (1) supervision target, and (2) input concatenated with or replaced from given data.

Our first research question is – **RQ1: *"How to utilize explanation in CTR?"*.** Following Hase and Bansal (2021), we categorize code explanation into two groups, namely G1 and G2:

- G1: Give a model to an additional **supervision objective**, expecting that the model learns to focus on which part of a given input data.
  - Regularization (REG): constrain model where to focus on from input data.
  - Generation (GEN): generate explanation for better task understanding.
- G2: Feed explanation to model as **augmented or replaced input**, allowing the model to utilize useful information from the explanation.
  - Concatenation (CAT): add the explanation to the given input data.
  - Replacement (RPL): replace the given input with its explanation.

G1 is adding a new target task of using explanation as model's objective. Here, the target task can be either REG or GEN. For REG, explanation contains where to focus on from input data, e.g., by highlighting the specific part(s) of the input data. The training model is now supervised to focus on these highlighted input features via attention supervision as in Pruthi et al. (2020); Stacey et al. (2021); Liu and Avci (2019); Small et al. (2011). For GEN, the model takes an input data (with its label) and produce its explanation as in Srivastava et al. (2018); Narang et al. (2020); Camburu et al. (2018); Hase et al. (2020). By doing so, we hope the model better understands the target task, resulting performance improvement.

G2 directly uses explanation as model's input. In this case, we can either concatenate (CAT) explanation with the given input or replace it (RPL). CAT is common in Natural Language Inference task, a general form of explanation is to supplement of input Camburu et al. (2018), e.g., given a premise "A child in a yellow plastic safety swing is laughing as a dark-haired woman in pink and coral pants stands behind her." and a hypothesis "A young mother is playing with her daughter in a swing." with the true label "neutral", the explanation is "Child does not imply daughter and woman does not imply mother." In other case, if explanation was self-contained, we could remove the given input and feed only the exaplantion (RPL). This can be viewed as an ideal form of the regularization method, as only the label related features are taken into the model. In CTR, explanation is self-contained since a pseudocode is an abstractive form of its real code.

To compare the effectiveness of each method, we build four baselines of REG, GEN, CAT, and RPL. We conduct code retrieval task on NAPS Zavershynskyi et al. (2018)

dataset, which contains (code, text, pseudocode) triplets. The experimental results showed that RPL approach is the best choice for CTR. However, a core problem is that explanation for code is expensive. Furthermore, the second group, including the winning approach RPL, requires explanation in test time.

Thus our second research question is – **RQ2:** ***"How to automatically generate explanation in CTR?"***. To this end, we applied two types of automated explanation generation methods: 1) Training Code-to-Pseudo (C2P) model and 2) Rule-based generation. Our finding is that, with RPL method, Rule-based automated explanation generation is a good way to improve CTR learning, showing 17.6%p of MRR improvement compared to without using explanation on NAPS code search. However, in scenarios, where rules are yet to be defined or cannot be reverse-engineered, we can see that C2P has an unrealized potential to perform comparably to Rule-based, while we leave such realization as future directions.

## 2. Related Work

### 2.1. Explanation

Our work can be viewed as providing explanation (pseudocode) to machine models, to improve traditional supervised learning. Existing uses of explanations on machine models can be categorized into two groups, according to Hase and Bansal (2021): As supervision target and as input. The first category generates additional supervision target. With additional highlights from human annotator, Zaidan et al. (2007) provide an auxiliary task of contrasting the original input and the highlight-masked input. Pruthi et al. (2020) constrain models to focus on highlighted features via multi-task learning and the winning approach, attention regularization. Stacey et al. (2021) extract highlights from a free form of text explanation and give attention supervision. Liu and Avci (2019) add L2 loss between unintended bias and task-specific prior features to unbias toxic terms, improving performance on text classification tasks under data-scarce setting. Small et al. (2011) allow annotators to provide ranked features toward labels, used as constraints. Zhang et al. (2016) use annotated rationales to encode the likelihood of sentences' being neutrals/rationales in a document. Srivastava et al. (2018) parse human explanations to constrain models for zero-shot learning. Other works generate explanations directly: In some approaches, such as in Narang et al. (2020), the task model tries to generate explanation as well. Camburu et al. (2018) conduct various experiments using human annotated explanations including predicting a label then generating an explanation for the predicted label, and generating an explanation then predicting the label with that generated explanation only. Hase et al. (2020) penalize label leakage during explanation generation to improve the quality of generated explanations.

The second category assumes explanation is given as additional input, which, together with input, helps the model predict labels better. Hase and Bansal (2021) empirically confirm that using explanation as input is better than as target. Lei Ba et al. (2015); Andreas et al. (2017); Rupprecht et al. (2018) take text description to generate class weights of latent vectors from an image encoder. Co-Reyes et al. (2018) utilize human-in-the-loop of receiving explanation from human experts as additional input.

### 2.2. Automated Explanation Generation

In order not to require human annotators to provide all explanation for training, an explanation generation method would be trained instead. Pruthi et al. (2020) automatically highlight salient features. Zhang et al. (2016) generate rationales via probabilities encoding the likelihood over labels. Andreas et al. (2017) retrieve a test time description from the training dataset during test time. Zhou et al. (2020); Rajani et al. (2019); Wiegreffe et al. (2020) train explanation generation from given input and label on (input, explanation, label) triplets, then augment explanation on (input, label) pairs.

### 2.3. Code-Text Representation Learning

To learn CTR, CodeBERT Feng et al. (2020) proposes a BERT based model of which pretraining strategy follows that of ELECTRA Clark et al. (2020). Further, GraphCodeBERT Guo et al. (2020) concatenates data flow with the code. Meanwhile, PLBART Ahmad et al. (2021) uses a BART Lewis et al. (2019) based model to pretrain both an encoder and a decoder. As these approaches focus on adding more information on the code modality, they are orthogonal to our approach matching cross-modalities between text and code.

## 3. Method

### 3.1. Preliminary

For code-text downstream tasks, such as code search with a text query $x$ and a candidate code $c$, a classic approach is supervising with $(x, c)$ pairs. Our target problem is exploring whether providing explanation $e$, helping a model to correctly predict answers, as loosely defined in Hase and Bansal (2021), can bridge the semantic gap between the two modalities of $x$ and $c$ for such training. A pseudocode is a human-readable form of the given code– an Intermediate Representation (IR) between code and text. We hope that using pseudocode as explanation $e$ allows CTR models to connect code and text (that are sharing the same semantics) in the latent space. To validate the effectiveness of learned Code-Text Representation, we select code search, which aims to match natural language queries of code descriptions with relevant codes, as a target task.

Formally, given a set of natural language code descriptions $X$, code corpus $C$, and explanations $E$, our objective is training a model $f$ with parameters $\theta$ to learn the relation $y \in \{0, 1\}$ between a code description $x$, a code $c \in C$, and its pseudocode $e \in E$ as an explanation:

$$\hat{\theta} \triangleq \operatorname*{argmin}_{\theta} L(f(x, c, e; \theta), y), \tag{1}$$

where $L$ is a loss function.

### 3.2. Solution Space of Leveraging Explanation

As Section 1 categorizes, we consider the following four groups: REG, GEN, CAT, and RPL.

### 3.2.1. Regularization (REG)

REG constrains model parameters to concentrate on causal features. Specifically, we adopt attention supervision to the attention in the model, as Pruthi et al. (2020) reported its effectiveness. The remaining question is how to obtain attention supervision. Following Pruthi et al. (2020); Stacey et al. (2021), we use token overlap between code $c$ and explanation $e = \{e_0, e_1, ...e_i, ..., e_{|e|}\}$, where $|e|$ is the length of $e$. Our distinction is that we weigh overlapped tokens to smooth the importance, using BM25 Robertson et al. (1995):

$$\alpha_{e_i} \triangleq \text{BM25}(c, e_i; C), \tag{2}$$

$$\hat{\alpha}_{e_i} = \frac{\alpha_{e_i}}{\Sigma_{k=1}^{|e|} \alpha_{e_k}}. \tag{3}$$

As we use a transformer-based model as $f$, i.e., CodeBERT Feng et al. (2020), there are 12 self-attention layers with 12 attention heads for each layer. Also, though self-attention is a 2-dimensional $N \times N$ matrix where $N$ is the input sequence length, our supervision is 1-dimensional vector of $N$ size. Thus, following Pruthi et al. (2020), the loss we give is KL divergence between $\alpha_\theta$, the attention weights of $[CLS]$ token on the last layer averaged across all attention heads, and $\hat{\alpha}_e$, the target attention weights from the explanation:

$$L_{reg} \triangleq -\lambda \ \text{KL}(\hat{\alpha}_e || \alpha_\theta), \tag{4}$$

where $\lambda$ is a hyperparameter.

### 3.2.2. Generation (GEN)

GEN adds a target objective to produce explanation, as discussed in Narang et al. (2020); Camburu et al. (2018). We use encoder-decoder structure for this purpose where the encoder is CodeBERT and the decoder is 6-layers transformer, following Code-to-Text implementation from Lu et al. (2021) where the same structure is used to generate comments or docstrings from code. We generate pseudocode instead of the text description. The loss used in this process is given as

$$L_{gen} \triangleq \Sigma_{i=0}^{|e|} \text{CE}(f(c, e_{<i}; \theta, \delta), \mathbb{1}_V(e_i)), \tag{5}$$

where CE means Cross Entropy loss, $\delta$ is the parameters of the decoder, $e_{<i}$ is an explanation sequence before time step $i$, i.e., a teacher-forced input, and $\mathbb{1}_V(e_i)$ is the target one-hot vector of $e_i$ over the vocabulary $V$. We pretrain the model with explanation generation task, then finetune on the target task Camburu et al. (2018).

### 3.2.3. Concatenation (CAT)

CAT is a general approach of using explanation: it attaches $e$ to $c$ and give $(x, c, e)$ as the model input Rajani et al. (2019); Wiegreffe et al. (2020); Zhou et al. (2020):

$$\hat{y} = f(x, c, e; \theta), \tag{6}$$

where $\hat{y}$ is the model prediction. The possible drawback of this method is that, since the length of explanation is added to the total input length, we may not be able to utilize the full input but truncate them. In CTR, real-world code, e.g., code in Github, often consists of hundreds to thousands of lines. Thus, code input solely can be longer than the max input length allowed by transformer-based models that we use, e.g., 512 tokens in CodeBERT.

#### 3.2.4. Replacement (RPL)

Instead of using $c$, RPL switches $c$ into $e$, then give $(x, e)$ as the model input Wiegreffe et al. (2020):

$$\hat{y} = f(x, e; \theta). \tag{7}$$

Compared to CAT, this is both efficient and effective in CTR, for that the explanation, pseudocode, is self-contained.

From the comparison results among above methods shown in Section 4.4.1, we confirmed that RPL is the optimal way of leveraging explanation.

### 3.3. Automated Explanation Generation

One key problem to use explanation is that a large number of labeled examples with human explanations are required. Moreover, some methods require such human explanations not only in training time but also in test time as well. In many cases, a large set of human annotations is prohibitively expensive. Therefore, we seek an approach that allows automated generation of explanation of decent quality, so that we could use those generated explanations 1) for data augmentation in training time or 2) as input explanation in test time. To tackle this, we propose two baselines: Code-to-Pseudo generation and Rule-based generation.

#### 3.3.1. Code-to-Pseudo Generation

Code-to-Pseudo (C2P) method generates an explanation, i.e., pseudocode, from its raw code. This is is a reversed approach of Pseudo-to-Code generation methods, which generates a raw code via its (human annotated) pseudocode Zhong et al. (2020); Zavershynskyi et al. (2018); Kulal et al. (2019):

$$e = \Pi_{i=0}^{|e|} f(c, e_{<i}; \theta, \delta). \tag{8}$$

Note that the loss function for C2P is the same as Equation (5). However, C2P itself is a hard task; C2P requires to transfer the code form into the natural language form, yet retain the semantics, e.g., the logic of the code.

#### 3.3.2. Rule-based Generation

While training a C2P model is not an easy task, we may adopt a simpler approach of converting a raw code into its pseudocode with predefined rules Zavershynskyi et al. (2018):

$$e = \phi(c; \mathfrak{R}), \tag{9}$$

where $\phi$ is a deterministic function following a set of predefined rules $\mathfrak{R}$, consisting of mappings between code phrases and its pseudocode descriptions. This approach can safely keep code semantics, while changing the form.

In Section 4.4.2, we first compare the above approaches 1) for data augmentation in training time, then confirm the effectiveness 2) as input explanation in test time, along with human annotated explanations. The results showed that Rule-based generation is suitable for both 1) and 2).

## 4. Experiments

### 4.1. Target Task

To validate CTR learning, we choose code search as the target task, which measures the semantic relatedness between text and code. Given a natural language code description $x$ and a code corpus $C$, the first objective is to predict whether a given pair $(x, c)$ is related so that $c$ answers $x$, and the second objective is to retrieve the most relevant code(s) $c \in C$ for $x$. We use the first objective for training a model, then use the trained model for the second objective at test time Feng et al. (2020); Guo et al. (2020); Lu et al. (2021). To measure the performance, we use widely used retrieval metrics: Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), Recall at 5 (R@5), and Precision at 5 (P@5).

### 4.2. Dataset

NAPS Zavershynskyi et al. (2018) is a collection of (code, pseudocode) pairs on codeforces.com, originally designated to Pseudo-to-Code generation task. One of the characteristics of this dataset is that all the variable and function names are normalized into the forms of `var#` and `func#`. As discussed in Lu et al. (2021), this normalization improves the quality of the dataset, as the task model would no more be able to use shortcuts by lexical overlaps.

We repurpose this dataset into code search task, which is one of the CTR learning tasks, privileged by using pseudocodes as explanations. From the URL link to each piece of code, we gathered its problem description, ensuring that every instance in our dataset would be a triplet of the form (text, code, pseudocode).

Originally, NAPS dataset consists of two types of training set – Training A which comes with rule-based pseudocode (relatively larger in size, 16,410 instances) and Training B which contains human authored pseudocode (2,231 instances) – plus one testing set with human annotated explanation(pseudocode).

As one can observe, such configuration allows us to directly use Training B and test set for our purpose regarding human annotated explanations. From this point on, we refer to Training B of NAPS dataset as simply **NAPS**. The problem is that NAPS dataset does not provide any testing set equipped with rule-based explanation, nor the authors publicly open the specific rules used to generate those. To remedy this, we divide NAPS Training A set into splits of 9:1. We use the former for training (with Training B) and the latter for simulating Rule-based test-time explanation generation. We call this setting, **NAPS+**.

### 4.3. Implementation

In all settings, we use a cross-encoder version of CodeBERT-base Feng et al. (2020) as a downstream task model, with max token length of 512, learning rate of 1e-5, batch size of 16, and an AdamW optimizer with an adam epsilon as 1e-8.

### 4.3.1. Explanation Utilization Methods

- NULL: This is the conventional approach of code search task that trains a model with $(x, c)$ pairs as input, without utilizing explanations.

- REG: We use attention supervision to the last layer's [CLS] token attention weights, averaging all attention heads. The target weights are derived from the explanation $e$ of the input code $c$ with the standard settings of BM25 Robertson et al. (1995). For the regularization hyperparameter $\lambda$, we use 1.

- GEN: From a given code $c$, this method generates its explanation $e$, then finetuned on the downstream task. We use NAPS for the generation task with a learning rate of 5e-5, a batch size of 80, a beam size of 10, both a source and target max length are 512, and an AdamW optimizer with an adam epsilon as 1e-8.

- CAT: Instead of feeding the model with concatenation of $x$ and $c$, this approach additionally appends $e$ at the end of the input sequence. When it comes to truncating the concatenated sequence $(x, c, e)$ if it exceeds the max length limit, we followed the truncation strategy below: (1) The shortest of the three gets 1/3 of the max length, at maximum; (2) In a similar fashion, the shorter of the remaining two can get up to 1/2 of the remaining length; and (3) The longest of the three utilizes the remaining slots.

- RPL: Instead of concatenating all the $(x, c, e)$ triplets, this method use $(x, e)$ pairs as input, due to an explanation – a pseudocode is self-contained and may not need information in $c$. The hyperparameter settings are the same as those in NULL.

### 4.3.2. Automated Explanation Generation Methods

- C2P: We train a Code-to-Pseudo model from the small collection of ($c$, human-written $e$) pairs, i.e., NAPS. Then, using C2P, we augment generated explanations of $(x, c)$ pairs to build NAPS+. For C2P, we use CodeBERT-base Feng et al. (2020) as the generation model with the same hyperparameters used in GEN.

- Rule-based: Using a set of predefined rules $\mathfrak{R}$ from Zavershynskyi et al. (2018), we translate each code $c$ into its explanation $e$ in $(x, c)$ pairs to build NAPS+. For each code phrase, there are more than one explanation phrases to be mapped, which alleviates overfitting to the specific explanation phrases.

After augmenting explanations in both cases, we train a code search model with $(x, e)$ pairs, i.e., using RPL method.

## 4.4. Results

As we claimed two research questions in Section 1, we answer each of them with the experimental results.

Table 1: Comparison of various explanation utilization methods for code search using human explanation from NAPS.

| TYPE | METHOD | MRR | MAP | P@5 | R@5 |
|---|---|---|---|---|---|
| w/o explanation | NULL | 0.0263 | 0.0192 | 0.0060 | 0.0032 |
| Explanation as target | REG | 0.0248 | 0.0199 | 0.0030 | 0.0093 |
| | GEN | 0.0364 | 0.0234 | 0.0134 | **0.0269** |
| Explanation as input | CAT | 0.0221 | 0.0208 | 0.0060 | 0.0108 |
| | RPL | **0.0489** | **0.0346** | **0.0164** | 0.0170 |

Table 2: Comparison of automated explanation generation methods over code search task on NAPS+.

(a) NAPS+ with **human** testing

| METHOD | MRR | MAP | P@5 | R@5 |
|---|---|---|---|---|
| NULL | 0.0880 | 0.0707 | 0.0328 | 0.0517 |
| C2P explanation | 0.0514 | 0.0330 | 0.0134 | 0.0241 |
| Rule-based explanation | **0.2485** | **0.1976** | **0.1090** | **0.2082** |

(b) NAPS+ with **gen** testing

| METHOD | MRR | MAP | P@5 | R@5 |
|---|---|---|---|---|
| NULL | 0.0803 | 0.0435 | 0.0301 | 0.0166 |
| C2P explanation | 0.0453 | 0.0180 | 0.0142 | 0.0040 |
| Rule-based explanation | **0.2563** | **0.1609** | **0.1593** | **0.0796** |

#### 4.4.1. **RQ1: *"How to utilize explanation in CTR?"***

Table 1 reports our results for NAPS: Only GEN and RPL outperformed NULL, while **RPL significantly outperforms all others**. We thus fix to use RPL, for the remaining evaluations.

#### 4.4.2. **RQ2: *"How to automatically generate explanation in CTR?"***

Table 2 evaluates diverse automated explanation generation methods. First, we validated the automated explanation generation methods in NAPS+ shown in Table 2a. Compared to Table 1, the accuracy of NULL was improved from having a bigger training instances. Rule-based method significantly outperformed both NULL and C2P, while C2P performs worse

Table 3: Comparison of rule-based explanation method with pseudocode retrieved with Code-to-Pseudo search model, i.e., C2P$_{enc}$, on NAPS+ with rule testing.

| METHOD | MRR | MAP | P@5 | R@5 |
|---|---|---|---|---|
| Rule-based explanation | 0.2563 | **0.1609** | **0.1593** | **0.0796** |
| C2P$_{enc}$ explanation | **0.2648** | 0.1492 | 0.1381 | 0.0736 |

than NULL. This signifies that the quality of generated explanations determines accuracy, as we further analyze this in Section 4.5.

Second, when explanation is required as input, it is burdensome to provide at test time. Table 2b shows the results, for automatically generated explanation at test time – Consistently to Table 2a, Rule-based explanation showed the best performance, validating the effectiveness of using Rule-based explanations at test time. Though Table 2a and 2b are not directly comparable, we can see values lie in a comparable range and observations are consistent.

#### 4.4.3. On unrealized potential of C2P

While Rule-based, a non-parametric approach outperforms the parametric approach of C2P at this point, C2P still has an advantage that it does not require to define rules, which are non-trivial and are specific to programming languages. More often, such rules are yet to be defined, or not known, as in our evaluation scenarios. Recall that, in our scenarios, the transformed result of applying rules were available, but not the rules themselves. Also, we need to identify the cause of unsatisfactory performance of C2P, so that we focus on our future efforts to improve this line of approach. Our hypothesis is that C2P is effective as an encoder, and can function well, if a closed set of explanations can be defined. In ExpBERT Murty et al. (2020), there is a "global" set of explanations, from which we can select a close match, in place of decoders. To validate this hypothesis, we build a Code-to-Pseudo retrieval model, which we denote as C2P$_{enc}$. For training C2P$_{enc}$, we use (code, rule-based pseudo) pairs, allowing the model to learn the rules latently. On the inference time of code search, for each code candidate, we use C2P$_{enc}$ to retrieve the relevant pseudocode from the pseudo collections, instead of directly using its rule-based pseudocode. Table 3 shows that C2P$_{enc}$ was comparable with Rule-based, indicating C2P has an unrealized potential that possibly eliminates the need to define rules. Guiding decoder with prior knowledge, to follow syntax and preserve semantics, is an important step to overcome to realize such potential.

### 4.5. Analysis

Table 4 shows different types of real examples: a text, a code, and two explanations generated by C2P and Rule-based. From the examples, we can see two points: First, the difference between the text and code modalities are significant, barely sharing the tokens. This ex-

Table 4: Examples of each data type: text, code, *C2P*, and *Rule-based*. Note that 1) *C2P* and *Rule-based* examples – explanations have intermediate modality between text and code examples, and 2) *C2P* breaks code semantics whereas *Rule-based* keeps.

| TYPE | EXAMPLE |
|---|---|
| Text | Vasily has a number a , which he wants to turn into a number b . For this purpose , he can do two types of operations : You need to help Vasily to transform the number a into the number b using only the operations described above , or find that it is impossible . Note that in this task you are not required to minimize the number of operations . It suffices to find any way to transform a into b . |
| Code | ( ( func ___main___ char * * [ ( var0 int ) ( var1 int ) ] [ ( var2 int * ) ( var3 bool ) ( var4 int ) ( var5 char * * ) ] [ ( = var5 ( _ctor char * * ) ) ( = var2 ( _ctor int * ) ) ( = var3 False ) ( array_push var2 var1 ) ( while True [ ( if ( < var1 var0 ) [ ( break ) ] [ ] ) ( if ( == var1 var0 ) [ ( = var3 True ) ( break ) ] [ ] ) ( if ( == ( % var1 10 ) 1 ) [ ( = var1 ( / var1 10 ) ) ] [ ( if ( == ( % var1 2 ) 0 ) [ ( = var1 ( / var1 2 ) ) ] [ ( break ) ] ) ] ) ( array_push var2 var1 ) ] [ ] ) ( if ( ! var3 ) [ ( = var5 ( array_concat var5 ( string_split " NO " " \|\| t " ) ) ) ] [ ( = var5 ( array_concat var5 ( string_split " YES " " \|\| t " ) ) ) ( array_push var5 ( str ( len var2 ) ) ) ( = var2 ( reverse var2 ) ) ( foreach var4 var2 [ ( = var5 ( array_concat var5 ( string_split ( concat ( str var4 ) " \|\| " ) " \|\| t " ) ) ) ] ) ] ) ( return var5 ) ] ) ) |
| C2P | given integers var0 , var1 and an array var2 . initialize var3 to the length of var1 , var4 to 0 . for each var5 from 0 to var2 exclusive , perform the following operations . if var1 equals 0 , set var2 to the following operations . add var1 to the end of var2 . if var1 contains the end of var1 contains the end of var1 then return var2 . |
| Rule-based | given integers var0 and var1 . var5 is an empty array of strings . var2 is an empty array of integers . initialize var3 to false . append var1 to the end of var2 . while true stop iterating if var1 is smaller than var0 , set var3 to true , stop iterating if var1 is equal to var0 , remove last digit from var1 if last digit of var1 is equal to one otherwise divide var1 by 2 if var1 is even , otherwise stop iterating , append var1 to the end of var2 . if not var3 add NO to var5 , and add YES to var5 , append string representation of the length of var2 to the end of var5 , you have to set var2 to reverse of var2 , for every var4 in var2 add concatenation of string value of var4 and space to var5 otherwise . you have to return var5 |

plains why using explanations is helpful; As explanations – C2P and Rule-based examples have intermediate modalities between text and code. Second, C2P breaks code semantics whereas Rule-based keeps. C2P contains wrong assignments, using an undefined variable, and returning a meaningless value. This makes an understanding of why there are significant performance gaps between C2P and Rule-based, throughout Table 2a and Table 2b. In other words, this informs that if we advance C2P to ensure sustaining code semantics in explanations, C2P, which is a deep learning approach, may outperform Rule-based.

## 5. Conclusion

In this paper, we studied the problem of bridging the gap in Code-Text Representation (CTR) learning, using the intermediate representation as an explanation. Inspired by a rich body of literature, leveraging explanation for model training, we identify a solution space of four approaches REG, GEN, CAT, and the winning approach RPL. Meanwhile, as explanation requires human efforts, we applied two baselines of automated explanation generation methods. The experimental results showed that Rule-based explanation generation method significantly improved a CTR related task, i.e., code search. Furthermore, we speculate that C2P has a potential to outperform Rule-based, which will be solved in future work. To the best of our knowledge, this work is the first step to define and categorize explanation in CTR learning.

## References

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. *arXiv preprint arXiv:2103.06333*, 2021.

Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. *arXiv preprint arXiv:1711.00482*, 2017.

Oana-Maria Camburu, Tim Rocktäschel, Thomas Lukasiewicz, and Phil Blunsom. e-snli: Natural language inference with natural language explanations. *arXiv preprint arXiv:1812.01193*, 2018.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, Jacob Andreas, John DeNero, Pieter Abbeel, and Sergey Levine. Guiding policies with language via meta-learning. *arXiv preprint arXiv:1811.07882*, 2018.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

Peter Hase and Mohit Bansal. When can models learn from explanations? a formal framework for understanding the roles of explanation data. *arXiv preprint arXiv:2102.02201*, 2021.

Peter Hase, Shiyue Zhang, Harry Xie, and Mohit Bansal. Leakage-adjusted simulatability: Can models generate non-trivial explanations of their behavior in natural language? *arXiv preprint arXiv:2010.04119*, 2020.

Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S Liang. Spoc: Search-based pseudocode to code. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/7298332f04ac004a0ca44cc69ecf6f6b-Paper.pdf.

Jimmy Lei Ba, Kevin Swersky, Sanja Fidler, et al. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4247–4255, 2015.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

Frederick Liu and Besim Avci. Incorporating priors with feature attribution on text classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6274–6283, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1631. URL https://www.aclweb.org/anthology/P19-1631.

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. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664, 2021.

Shikhar Murty, Pang Wei Koh, and Percy Liang. ExpBERT: Representation engineering with natural language explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2106–2113, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.190. URL https://aclanthology.org/2020.acl-main.190.

Sharan Narang, Colin Raffel, Katherine Lee, Adam Roberts, Noah Fiedel, and Karishma Malkan. Wt5?! training text-to-text models to explain their predictions. *arXiv preprint arXiv:2004.14546*, 2020.

Danish Pruthi, Bhuwan Dhingra, Livio Baldini Soares, Michael Collins, Zachary C Lipton, Graham Neubig, and William W Cohen. Evaluating explanations: How much do explanations from the teacher aid students? *arXiv preprint arXiv:2012.00893*, 2020.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*, 2019.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

Christian Rupprecht, Iro Laina, Nassir Navab, Gregory D Hager, and Federico Tombari. Guide me: Interacting with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8551–8561, 2018.

Kevin Small, Byron C Wallace, Carla E Brodley, and Thomas A Trikalinos. The constrained weight space svm: learning with ranked features. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 865–872, 2011.

Shashank Srivastava, Igor Labutov, and Tom Mitchell. Zero-shot learning of classifiers from natural language quantification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 306–316, 2018.

Joe Stacey, Yonatan Belinkov, and Marek Rei. Natural language inference with a human touch: Using human explanations to guide model attention. *arXiv preprint arXiv:2104.08142*, 2021.

Sarah Wiegreffe, Ana Marasovic, and Noah A Smith. Measuring association between labels and free-text rationales. *arXiv preprint arXiv:2010.12762*, 2020.

Omar Zaidan, Jason Eisner, and Christine Piatko. Using "annotator rationales" to improve machine learning for text categorization. In *Human language technologies 2007: The conference of the North American chapter of the association for computational linguistics; proceedings of the main conference*, pages 260–267, 2007.

Maksym Zavershynskyi, Alex Skidanov, and Illia Polosukhin. Naps: Natural program synthesis dataset. *arXiv preprint arXiv:1807.03168*, 2018.

Ye Zhang, Iain Marshall, and Byron C Wallace. Rationale-augmented convolutional neural networks for text classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing. Conference on Empirical Methods in Natural Language Processing*, volume 2016, page 795. NIH Public Access, 2016.

Ruiqi Zhong, Mitchell Stern, and Dan Klein. Semantic scaffolds for pseudocode-to-code generation. *arXiv preprint arXiv:2005.05927*, 2020.

Wangchunshu Zhou, Jinyi Hu, Hanlin Zhang, Xiaodan Liang, Maosong Sun, Chenyan Xiong, and Jian Tang. Towards interpretable natural language understanding with explanations as latent variables. *arXiv preprint arXiv:2011.05268*, 2020.

## Appendix A. Full result table

Table 5: Full comparison of automated explanation generation methods over code search task on NAPS+.

(a) NAPS+ with **human** testing

| Training A | Training B | TEST INPUT | MRR | MAP | P@5 | R@5 |
|---|---|---|---|---|---|---|
| - | Code | Code | 0.0263 | 0.0192 | 0.0060 | 0.0032 |
| | | C2P | 0.0442 | 0.0269 | 0.0104 | 0.0125 |
| | | Human | 0.0441 | 0.0211 | 0.0119 | 0.0075 |
| - | Human explanation | Code | 0.0380 | 0.0319 | 0.0090 | 0.0193 |
| | | C2P | 0.0445 | 0.0262 | 0.0090 | 0.0115 |
| | | Human | 0.0489 | 0.0346 | 0.0164 | 0.0170 |
| Code | Code | Code | 0.0880 | 0.0707 | 0.0328 | 0.0517 |
| | | C2P | 0.0705 | 0.0447 | 0.0179 | 0.0404 |
| | | Human | 0.0526 | 0.0354 | 0.0164 | 0.0414 |
| C2P explanation | Human explanation | Code | 0.0209 | 0.0172 | 0.0030 | 0.0017 |
| | | C2P | 0.0350 | 0.0269 | 0.0075 | 0.0068 |
| | | Human | 0.0514 | 0.0330 | 0.0134 | 0.0241 |
| Rule-based explanation | Human explanation | Code | 0.0991 | 0.0773 | 0.0493 | 0.0818 |
| | | C2P | 0.0728 | 0.0435 | 0.0299 | 0.0334 |
| | | Human | **0.2485** | **0.1976** | **0.1090** | **0.2082** |

(b) NAPS+ with **gen** testing

| Training A | Training B | TEST INPUT | MRR | MAP | P@5 | R@5 |
|---|---|---|---|---|---|---|
| - | Code | Code | 0.0629 | 0.0190 | 0.0159 | 0.0083 |
| | | C2P | 0.0156 | 0.0350 | 0.0124 | 0.0041 |
| | | Rule-based | 0.0385 | 0.0159 | 0.0053 | 0.0013 |
| - | Human explanation | Code | 0.0370 | 0.0177 | 0.0088 | 0.0024 |
| | | C2P | 0.0254 | 0.0415 | 0.0124 | 0.0203 |
| | | Rule-based | 0.0504 | 0.0186 | 0.0124 | 0.0032 |
| Code | Code | Code | 0.0803 | 0.0435 | 0.0301 | 0.0166 |
| | | C2P | 0.0045 | 0.0251 | 0.0000 | 0.0000 |
| | | Rule-based | 0.0623 | 0.0253 | 0.0195 | 0.0061 |
| C2P explanation | Human explanation | Code | 0.0503 | 0.0183 | 0.0142 | 0.0042 |
| | | C2P | 0.0144 | 0.0331 | 0.0124 | 0.0034 |
| | | Rule-based | 0.0453 | 0.0180 | 0.0142 | 0.0040 |
| Rule-based explanation | Human explanation | Code | 0.1272 | 0.0774 | 0.0761 | 0.0324 |
| | | C2P | 0.0214 | 0.0373 | 0.0177 | 0.0115 |
| | | Rule-based | **0.2563** | **0.1609** | **0.1593** | **0.0796** |

## Appendix B. Analysis

Table 6: Examples of each data type: text, code, C2P, and human.

| TYPE | EXAMPLE |
|---|---|
| Text | Ilya is a very good-natured lion. He likes maths. Of all mathematical objects, his favourite one is matrices. Now he's faced a complicated matrix problem he needs to solve. He's got a square 2n×2n-sized matrix and 4n integers. You need to arrange all these numbers in the matrix (put each number in a single individual cell) so that the beauty of the resulting matrix with numbers is maximum. The beauty of a 2n×2n-sized matrix is an integer, obtained by the following algorithm: As you can see, the algorithm is recursive. Help Ilya, solve the problem and print the resulting maximum beauty of the matrix. |
| Code | ( ( func ___main___ int [ ( var0 int * ) ] [ ( var1 int ) ( var2 int * ) ( var3 int ) ( var4 int ) ( var5 int ) ( var6 int ) ( var7 int ) ( var8 int ) ] [ ( = var8 0 ) ( = var1 ( len var0 ) ) ( = var2 ( _ctor int * var1 ) ) ( = var3 0 ) ( while ( < var3 var1 ) [ ( = ( array_index var2 var3 ) ( array_index var0 ( - ( = var8 ( + var8 1 ) ) 1 ) ) ) ] [ ( = var3 ( + var3 1 ) ) ] ) ( = var2 ( sort var2 ) ) ( = var4 0 ) ( = var5 1 ) ( while ( <= var5 var1 ) [ ( = var6 var5 ) ( = var7 ( - var1 1 ) ) ( while ( > var6 0 ) [ ( = var4 ( + var4 ( array_index var2 var7 ) ) ) ] [ ( = var7 ( - var7 1 ) ) ( = var6 ( - var6 1 ) ) ] ) ] [ ( = var5 ( * var5 4 ) ) ] ) ( return var4 ) ] ) ) |
| C2P | you are given an array of numbers var0 ( indexing is 0 - based ) . you have to create an array of numbers var2 ( indexing is 0 - based ) . you have to set var5 to zero and var6 to zero . for each position var6 in var2 you have to add var2 [ var6 ] to var5 if var5 is greater than or equal to var2 [ var6 ] you have to add 1 to var6 . you have to return var5 . |
| Human | given an integer array var0 , let var1 be the size of var0 . create an array var2 identical to var0 . sort var2 . initialize var4 to 0 . set var5 to 1 , so long as var5 is < = var1 , do the following in a loop : set var6 to var5 . initialize var7 to var1 - 1 , do the following in a loop so long as var6 is more than 0 : add var2 [ var7 ] to var4 , and decrement var7 and var6 . ( end of inner loop ) set var5 to var5 * 4 . ( end of outer loop ) return var4 . |