Towards Generating Summaries for Lexically Confusing Code through Code Erosion

Fan Yan and **Ming Li** [*]
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{yanf, lim}@lamda.nju.edu.cn

Abstract

Code summarization aims to summarize code functionality as high-level nature language descriptions to assist in code comprehension. Recent approaches in this field mainly focus on generating summaries for code with precise identifier names, in which meaningful words can be found indicating code functionality. When faced with lexically confusing code, current approaches are likely to fail since the correlation between code lexical tokens and summaries is scarce. To tackle this problem, we propose a novel summarization framework named VECOS. VECOS introduces an *erosion* mechanism to conquer the model's reliance on precisely defined lexical information. To facilitate learning the eroded code's functionality, we force the representation of the eroded code to align with the representation of its original counterpart via variational inference. Experimental results show that our approach outperforms the state-of-the-art approaches to generate coherent and reliable summaries for various lexically confusing code.

1 Introduction

Code summaries, also called code comments that summarize code functionality in natural language, are essential software components assisting in program comprehension. Due to developing time constraints, developers cannot write useful summaries for every code snippet, which leads to strong demand for automatic source code summarization.

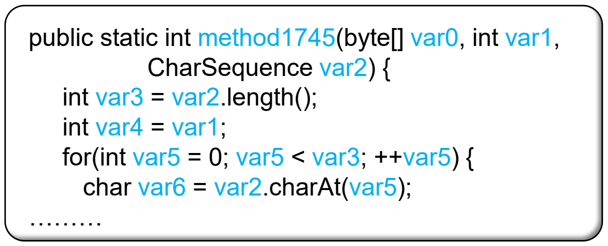
As a hot topic in software engineering, many research works have been conducted for automatic code summarization. Conventional approaches mainly rely on manually crafted rules [Moreno *et al.*, 2013] and information retrieval (IR) [Wong *et al.*, 2013]. With the development of neural machine translation, more researchers try to model code summarization as a translation from code to natural language. CodeNN [Iyer *et al.*, 2016] and CodeAttention [Zheng *et al.*, 2019] utilize attention LSTM for code encoding and generating summaries. Approaches like SBT [Hu *et al.*, 2018] attempt to extract structural semantics by taking abstract syntax tree as input. The aforementioned approaches, although effective, have a reliance on the correlation between code identifier tokens and summary words. To generate meaningful summaries, meaningful tokens must be observed in the code, which requires the code to be lexically precise.

However, code is not always lexically precise with well-defined identifier names. Firstly, source code files in the real-world are often obfuscated in which identifier names are replaced with short, opaque, and meaningless symbols (e.g. Figure 1(a)). Such obfuscation maintains the program semantics but diminishes code readability to hide business logic and prevent software plagiarism [Bavishi *et al.*, 2018; Tran *et al.*, 2019]. Secondly, it is nontrivial for developers, especially inexperienced developers, to choose appropriate identifier names (e.g. Figure 1(b)). What is worse, identifier names are subject to decay during software evolution [Deissenboeck and Pizka, 2006]. The concept they refer to may be altered or abandoned during the software evolution, which may mislead the summarization model to extract incorrect semantics. Such lexically confusing code places significant challenges for current approaches since the correlation between code tokens and summary words becomes implicit. What makes the challenge bigger is that lexically confusing code has few useful human-written summaries [Deissenboeck and Pizka, 2006], which causes a lack of data to learn to adapt to those circumstances.

One question arises here: how can we generate high-quality summaries for lexically confusing code? One straightforward way is to replace all the identifier names with a single '`<OTHER>`' token and learns to map the replaced code to summaries [LeClair *et al.*, 2019]. Such an approach can reduce the model's dependence on user-defined identifier names. However, collapsing all the identifiers to *only one* token changes the functional semantics without alleviating the difficulty in essence, which yields poor evaluation performance. Intuitively, every lexically confusing code snippet is like an *eroded* version of a lexically precise one, in which the functional semantics remains the same but the lexical semantics are eroded. Suppose that the model can learn the erosion process explicitly and restore the eroded semantics in a latent feature space, it is possible to generate reliable summaries even without precise lexical information.

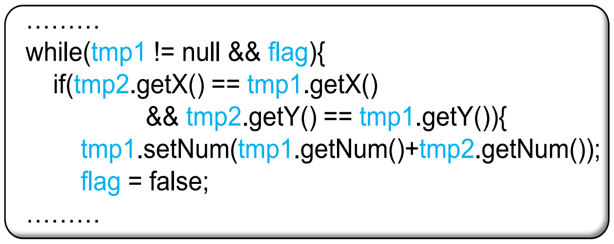
This paper proposes a novel code summarization framework called VECOS (Variational Eroded COde Summariza-

[*]Ming Li is the corresponding author.



```
public static int method1745(byte[] var0, int var1,
                CharSequence var2) {
    int var3 = var2.length();
    int var4 = var1;
    for(int var5 = 0; var5 < var3; ++var5) {
      char var6 = var2.charAt(var5);
………
```

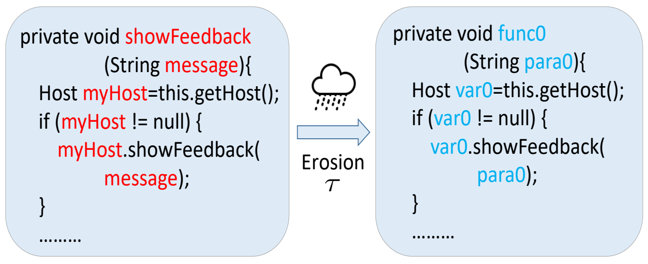
(a) An obfuscated code snippet from a reverse engineering project whose identifiers are replaced with meaningless symbols.



```
………
while(tmp1 != null && flag){
   if(tmp2.getX() == tmp1.getX()
              && tmp2.getY() == tmp1.getY()){
      tmp1.setNum(tmp1.getNum()+tmp2.getNum());
      flag = false;
………
```

(b) Poorly written loop code with inappropriate identifier names whose lexical semantics is deficient.

Figure 1: Lexically confusing code examples.



```
private void showFeedback
          (String message){
  Host myHost=this.getHost();
  if (myHost != null) {
    myHost.showFeedback(
          message);
  }
  ………
```

```
private void func0
          (String para0){
  Host var0=this.getHost();
  if (var0 != null) {
    var0.showFeedback(
          para0);
  }
  ………
```

Figure 2: An example of code erosion τ: the left part is the original code, the right part is the eroded code with user-defined identifier names replaced but code functionality remained.

tion) for lexically confusing code summarization. VECOS first simulates the lexical erosion process by converting every code snippet to a unified *eroded* version to adapt to various confusing identifier naming styles, in which the original identifiers are replaced with some pre-defined symbols. To learn the erosion process explicitly, we force the learned semantics representation of the eroded code to align with its lexically precise counterpart by variational reference. By doing that, VECOS learns how to "*de*-erode" code by restoring the eroded lexical semantics in a latent feature space. Experimental results show that VECOS can generate high-quality summaries for various lexically confusing circumstances.

The main contribution of this paper lies in two folds:

- We propose an erosion mechanism for code summarization to conquer the model's reliance on precise lexical information, and hence the problem of summarizing lexically confusing code can be formalized as learning from lexically precise code with an erosion process.
- We design a novel generative code summarization framework named VECOS for lexically confusing code summarization.

The rest of this paper is organized as follows: the second section explains our approach VECOS in detail. The third section shows the experiments together with some analysis. The fourth section introduces some related work. Finally, we conclude the whole paper.

2 Our Approach: VECOS

Let $\mathcal{D} = \{(\mathbf{c}_1, \mathbf{y}_1), (\mathbf{c}_2, \mathbf{y}_2), ..., (\mathbf{c}_n, \mathbf{y}_n)\}$ denotes a lexically precise code-summary corpus, where $\mathbf{c}_i$ stands for a code snippet whose identifiers are well specified and $\mathbf{y}_i$ stands for the related summary. Let $\mathcal{M} = (\tau, \psi)$ represents our summarization framework VECOS, where τ is a rule-based erosion function which maps $\mathbf{c}_i$ to its unified eroded version: $\widetilde{\mathbf{c}}_i = \tau(\mathbf{c}_i)$, and ψ is a tree-based neural summarization model generating code summaries $\widetilde{\mathbf{y}}'_i = \psi(\widetilde{\mathbf{c}}_i)$, where the generated summary is expected to be similar to the human-written one: $\widetilde{\mathbf{y}}'_i \simeq \mathbf{y}_i$. The training goal is, for any test case $\mathbf{c}'_j$, no matter how confusing the lexical information of $\mathbf{c}'_j$ is, the generated summary $\widetilde{\mathbf{y}}'_j$ is always reliable.

Unlike existing approaches, we take a code erosion process τ as the first step for code summarization. τ replaces user-defined identifiers with pre-defined symbols without changing functionality: $\widetilde{\mathbf{c}}_i = \tau(\mathbf{c}_i)$. For the training phase, such replacement forces the summarization model to learn semantics from the eroded code, cutting off the dependence on user-defined identifier tokens. For the testing phase, τ acts as a process of data standardization by converting various lexically confusing code to a unified eroded version, which is beneficial to improve the generalization ability.

We take the abstract syntax tree of eroded code $\widetilde{\mathbf{c}}_i$ as the input for ψ's encoder. We assume there exists a continuous latent feature space $\mathbf{Z}$ for code functionanlity representation, from which ψ infers the latent variable $\mathbf{z}_i$ with given $\widetilde{\mathbf{c}}_i$, i.e. $p_\psi(\mathbf{z}_i|\widetilde{\mathbf{c}}_i)$. Then $\mathbf{z}_i$ together with $\widetilde{\mathbf{c}}_i$ will be used to guide the summary decoding, i.e. $p(\mathbf{y}_i|\mathbf{z}_i, \widetilde{\mathbf{c}}_i)$. To learn the erosion process explicitly, we leverage the origin lexically precise code $\mathbf{c}_i$ and corresponding summary $\mathbf{y}_i$ to pre-train an auxiliary model ϕ. Since $\mathbf{c}_i$ and $\widetilde{\mathbf{c}}_i$ share the same functionality, we force the distribution $p_\psi(\mathbf{z}_i|\widetilde{\mathbf{c}}_i)$ inferred by ψ to assimilate $p_\phi(\mathbf{z}_i|\mathbf{c}_i)$ inferred by auxiliary model ϕ using variational inference. By doing that, the eroded semantics is expected to be restored in the latent feature space $\mathbf{Z}$.

The overall framework of VECOS is shown in Figure 3. An auxiliary model ϕ is first trained with the translation cross entropy loss. Since the input for ϕ is lexically precise, we take $p_\phi(\mathbf{z}_i|\mathbf{c}_i)$ as the lexical enriched prior of $p_\psi(\mathbf{z}_i|\widetilde{\mathbf{c}}_i)$. Our summarization model ψ is trained by exploiting both the translation supervision and the divergence between the prior and posterior distribution via variational inference.

2.1 Code Erosion

The code erosion part τ is responsible for converting arbitrary code snippets to a unified *eroded* version by replacing user-defined identifiers with ordered symbols. We introduce code

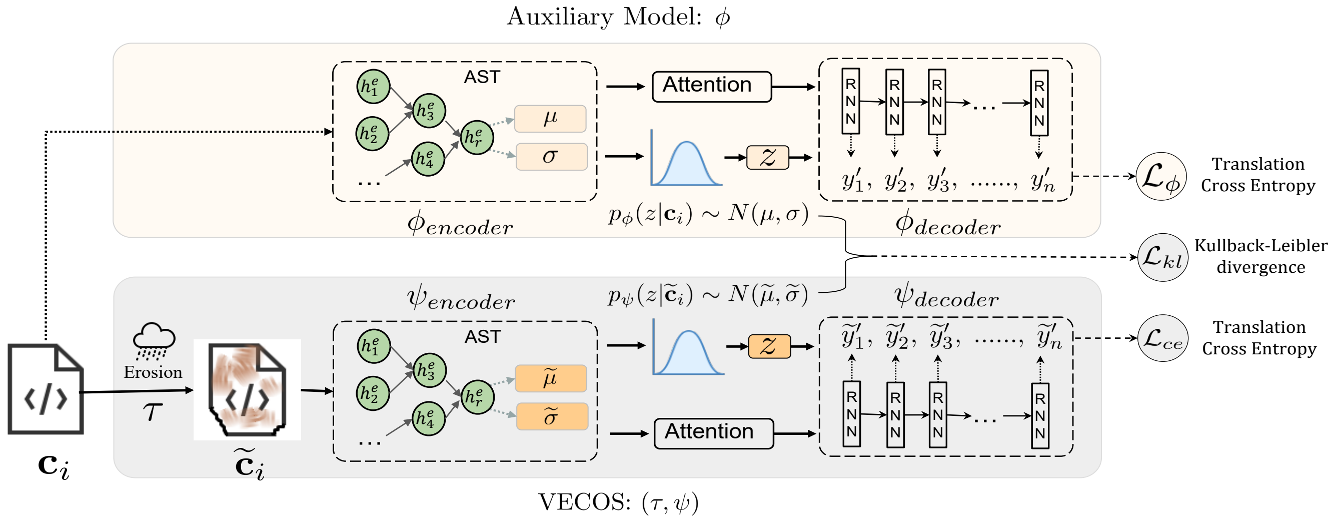

Auxiliary Model: ϕ
AST
Attention
μ
σ
z
$\phi_{encoder}$
$p_\phi(z|\mathbf{c}_i) \sim N(\mu, \sigma)$
$\phi_{decoder}$
$y'_1, y'_2, y'_3, \ldots\ldots, y'_n$
$\mathcal{L}_\phi$
Translation Cross Entropy
$\mathcal{L}_{kl}$
Kullback-Leibler divergence
$\mathcal{L}_{ce}$
Translation Cross Entropy
$\psi_{encoder}$
$p_\psi(z|\widetilde{\mathbf{c}}_i) \sim N(\widetilde{\mu}, \widetilde{\sigma})$
$\psi_{decoder}$
$\widetilde{y}'_1, \widetilde{y}'_2, \widetilde{y}'_3, \ldots\ldots, \widetilde{y}'_n$
Erosion
τ
$\mathbf{c}_i$
$\widetilde{\mathbf{c}}_i$
$\widetilde{\mu}$
$\widetilde{\sigma}$
RNN
VECOS: (τ, ψ)

Figure 3: The overall framework of our approach: VECOS.

obfuscation techniques to complete the erosion process τ. An example is provided in Figure 2. With function granularity, user-defined identifiers (the referenced external API names are not included) can be divided into function names, parameter names, and local variables. We take '`func`', '`para`', and '`var`' together with an order suffix to make the replacement. Our method preserves complete functional semantics and part of the identifier type information, which avoids ambiguity in functionality representation.

We design τ for two reasons. Firstly, most code snippets of the current code-summary corpus have well-defined identifiers correlated to code functionality. For example, for the Java code-summary corpus in [Hu *et al.*, 2018] and [Husain *et al.*, 2019], the identifier tokens can be found in the summary directly for more than 65% code-summary pairs. To prevent the model from relying on such lexical correlation, we introduce τ to conduct the erosion process to avoid exposing sufficient lexical information directly to the model. Secondly, the identifier naming of lexically confusing code is quite flexible since identifiers can be arbitrarily chosen and elude automated analysis. Various confusing naming styles can be unified as a single eroded version with the erosion process, which alleviates the summarization model's learning burden and enhances generalization towards unseen data.

2.2 Code Summarization

We propose a generative tree-based model ψ for code summarization. Previous discriminative neural summarization models usually rely on the attention mechanism to identify semantics alignment between code fragments and summary tokens. The functional semantics is learned implicitly. However, such alignment may deviate when the lexical correlation between the code and the summary is weak, which is common under lexically confusing circumstances. To tackle this problem, we introduce a latent variable z to represent code functional semantics explicitly. z is inferred from the given eroded code $\widetilde{\mathbf{c}}_i$, and used as a global semantics indicator for summarization. The encoder of ψ learns to approximate $p_\psi(z|\widetilde{\mathbf{c}}_i)$ while the decoder learns to approximate $p_\psi(\mathbf{y}_i|z, \widetilde{\mathbf{c}}_i)$. The generative process can be formulated as Equation 1:

$$p_\psi(\mathbf{y}_i|\widetilde{\mathbf{c}}_i) = \int_z p_\psi(\mathbf{y}_i|z, \widetilde{\mathbf{c}}_i) p_\psi(z|\widetilde{\mathbf{c}}_i) dz \tag{1}$$

Tree-based Encoder with Latent Variable

The tree-based encoder takes the abstract syntax tree (AST) of code as input. As shown in previous works [Hu *et al.*, 2018; Wan *et al.*, 2018; LeClair *et al.*, 2019], AST is capable of representing complex structural semantics of source code. We build our encoder based on Tree-LSTM [Tai *et al.*, 2015], which is a generalization of LSTM to tree-structured network topologies, traversing a single tree from leaf nodes to the root node. [Tai *et al.*, 2015] propose two kinds of Tree-LSTM: N-ary and Child-Sum. Considering the order of child nodes is necessary, we choose N-ary and set N as 2, in which N is the number of child nodes. Since traditional AST does not restrict the number of child nodes, we transform the original AST to a binary left-child-right-sibling tree.

Tree-LSTM outputs encoding for every node of AST, in which the encoding of the root node h^e_r is the synthesis of the whole tree. We parameterize the approximate posterior as $p_\psi(z|\widetilde{\mathbf{c}}_i) = \mathcal{N}(\mu, \sigma^2)$, in which the mean and variance vector is calculated from the encoding of the root node h^e_r:

$$\mu = tanh(W_\mu h^e_r + b_\mu); \quad \log \sigma^2 = tanh(W_\sigma h^e_r + b_\sigma) \tag{2}$$

In Equation 2, we introduce $tanh$ as the activation function. For one thing, $tanh$ increases the non-linearity of the model to improve the representation capability. For another, $tanh$'s lower bound prevents the value of σ from being so small that the model degenerates into a discriminative model. To ensure the gradient can be transferred to the encoder, z needs to be reparameterized using μ and σ:

$$z = \mu + \sigma \odot \epsilon \tag{3}$$

Equation 3 is the Reparameterize Technique, in which $\odot$ refers to the element-wise product and ϵ is a variable sampled from standard Gaussian distribution: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$.

Sequential Decoder with Attention

We take the vanilla sequential LSTM with the dot attention mechanism to decode the encoded information to code summaries. The decoding process can be formulated as:

$$\begin{aligned} h_i^d, s_i^d &= LSTM([x_i, c_{i-1}], h_{i-1}^d, s_{i-1}^d) \\ o_i &= Softmax(W_o \cdot [h_i^d, z] + b_o) \end{aligned} \tag{4}$$

In Equation 4, h_i^d, s_i^d are the hidden state and cell state of LSTM at time step i. o_i is the output word distribution of next generated summary token. Square brackets expression like $[h_i^d, z]$ means concatenation of two vectors. As shown in the last equation of 4, latent variable z is feed into the output layer together with current decoding hidden state h_i^d at every time step for generating summary tokens. c_{i-1} is the attentional weighted sum of encoders outputs: $c_{i-1} = \sum_{j=1}^{N} \alpha_{ij}^{\mathrm{T}} h_j^e$, in which α_{ij} is the attention score of previous decoding hidden state h_{i-1}^d and encoding output h_j^e: $\alpha_{ij} = \frac{\exp((h_j^e)^{\mathrm{T}} h_{i-1}^d)}{\sum_{j=1}^{N} \exp((h_j^e)^{\mathrm{T}} h_{i-1}^d)}$.

2.3 Training via Variational Alignment

Since the user-defined lexical information is eroded, it is non-trivial for the summarization model ψ to learn a well-formed semantics distribution of z for the eroded code $\widetilde{\mathbf{c}}_i$. Thus we propose to pre-train an auxiliary summarization model ϕ. ϕ shares the same network structure as ψ and is trained with the original lexically precise code corpus $\mathcal{D} = \{(\mathbf{c}_i, \mathbf{y}_i)\}$:

$$\mathcal{L}_\phi = \mathbb{E}_{(\mathbf{c}_i,\mathbf{y}_i)\sim\mathcal{D}} \left[\mathbb{E}_{z\sim p_\phi(z|\mathbf{c}_i)} \left[-\log p_\phi(\mathbf{y}_i|z, \mathbf{c}_i) \right] \right] \tag{5}$$

We train the auxiliary model ϕ by minimizing the cross-entropy loss $\mathcal{L}_\phi$. For one thing, ϕ can learn the underlying semantic distribution more effectively with $\mathbf{c}_i$ since it contains more lexical information than $\widetilde{\mathbf{c}}_i$: $p_\phi(z|\mathbf{c}_i) \simeq p(z|\mathbf{c}_i)$. For another, the erosion process τ does not change functionality. Thus the distribution of z should be the same given $\widetilde{\mathbf{c}}_i$ or given $\mathbf{c}_i$: $p(z|\widetilde{\mathbf{c}}_i) = p(z|\mathbf{c}_i)$. As a result, $p_\phi(z|\mathbf{c}_i)$ can be treated as a lexical-enriched prior guiding the learning of $p_\psi(z|\widetilde{\mathbf{c}}_i)$.

Inspired by stochastic gradient variational Bayes [Kingma and Welling, 2013], we propose a variational training framework to align $p_\psi(z|\widetilde{\mathbf{c}}_i)$ with $p_\phi(z|\mathbf{c}_i)$ by approximating maximum likelihood estimation of $\log p_\psi(\mathbf{y}_i|\widetilde{\mathbf{c}}_i)$:

$$\begin{aligned} \log p_\psi(\mathbf{y}_i|\widetilde{\mathbf{c}}_i) &\geq \mathbb{E}_{z\sim p_\psi(z|\widetilde{\mathbf{c}}_i)} \left[\log \left\{ \frac{p_\psi(\mathbf{y}_i, z|\widetilde{\mathbf{c}}_i)}{p_\psi(z|\widetilde{\mathbf{c}}_i)} \right\} \right] \\ &= \mathbb{E}_{z\sim p_\psi(z|\widetilde{\mathbf{c}}_i)} \left[\log p_\psi(\mathbf{y}_i|z, \widetilde{\mathbf{c}}_i) \right] \\ &\quad - KL \left[p_\psi(z|\widetilde{\mathbf{c}}_i) || p(z|\widetilde{\mathbf{c}}_i) \right] \end{aligned} \tag{6}$$

Equation 6 is the ELBO (Evidence Lower BOund) of the log likelihood. The encoder and decoder of ψ are used to model $p_\psi(z|\widetilde{\mathbf{c}}_i)$ and $p_\psi(\mathbf{y}_i|z, \widetilde{\mathbf{c}}_i)$ respectively. The first term of Equation 6 is the negative cross-entropy of generating target sequences $\mathbf{y}_i$. The second second term is the Kullback-Leibler(KL) divergence between learned $p_\psi(z|\widetilde{\mathbf{c}}_i)$ and its prior $p(z|\widetilde{\mathbf{c}}_i)$, which is assigned as $p_\phi(z|\mathbf{c}_i)$. For clarity, we use $\mathcal{L}_{ce}^{(i)}$ and $\mathcal{L}_{kl}^{(i)}$ to represent cross entropy and KL divergence for a given pair $(\widetilde{\mathbf{c}}_i, \mathbf{y}_i)$:

$$\mathcal{L}_{ce}^{(i)} = \mathbb{E}_{z\sim p_\psi(z|\widetilde{\mathbf{c}}_i)} \left[-\log p_\psi(\mathbf{y}_i|z, \widetilde{\mathbf{c}}_i) \right] \tag{7}$$

$$\mathcal{L}_{kl}^{(i)} = KL \left[p_\psi(z|\widetilde{\mathbf{c}}_i) || p_\phi(z|\mathbf{c}_i) \right] \tag{8}$$

The overall training objective for ψ is to minimize:

$$\mathcal{L}_\psi^{(i)} = \mathcal{L}_{ce}^{(i)} + \lambda_{kl} \cdot \mathcal{L}_{kl}^{(i)} \tag{9}$$

In Equation 9, we introduce a λ_{kl} to balance the translation loss and KL loss as a hyperparameter. Since variational encoder-decoder framework is tricky with RNN models, λ_{kl} is used to anneal the KL divergence to balance the value of the two terms [Bowman *et al.*, 2015; Bahuleyan *et al.*, 2018]. The training process of ψ can be regarded as a combination of a translation task by reducing cross entropy and distribution aligment by reducing KL divergence.

3 Experiment

To evaluate the effectiveness of VECOS, we conduct experiments on a benchmark dataset and compare it with several state-of-the-art approaches for code summarization.

3.1 Dataset Preparation

We take the same dataset as [Hu *et al.*, 2018], in which java code snippets are provided in functional granularity with corresponding summaries. For roughly 65% code summary pairs in this dataset, the identifier tokens can be found in the summary directly, which indicates the original identifiers are lexically precise to reveal code functionality. We do some filtering to this dataset. The class constructors, setters, getters, tester methods, and code snippets whose summaries are shorter than four words, are dropped. While our summarization model takes the tree-based data structure as input for the encoder, we cannot truncate it to a fixed size. So we filter out some extremely long code to avoid large zero padding. Code summaries vary in length. We only take the first sentence of a single summary and fix the length as 10. All the summary tokens are stemmed. Consequently, we obtain 274,302 code-summary pairs. We randomly split the dataset as 8:1:1 to construct datasets for training, validating, and testing. We provide two ways to replace original identifiers in the test set to simulate lexically confusing circumstances for testing:

- Symbolic replacement: well-defined code identifiers are replaced with ordered symbols such as '`var0`', '`var1`', which simulates obfuscated circumstances.
- Synonym replacement: sub-tokens of identifiers are replaced with synonyms. For example, '`upStreaming`' being replaced as '`upFlow`', in which 'streaming' and 'flow' are similar in general context but different in professional domains. Synonym replacement simulates inappropriate and flexible variable naming styles. The synonym is chosen by calculating the neighborhoods of word embeddings according to pre-trained GloVe embedding [Pennington *et al.*, 2014].

We replace identifiers of each test case according to varied ratios (0%, 25%, 50%, 75%, 100%), where 0% means no replacement so the code is still lexically precise and 100% means replacing every identifier. Both symbolic and synonym replacement will not change the functional semantics.

Model	0%	Symbolic Replacement 25%	50%	75%	100%	Synonym Replacement 25%	50%	75%	100%
CodeNN	18.33	13.43	11.06	8.12	7.82	12.17	10.61	8.27	7.97
AttnGRU	22.30	15.02	12.61	8.67	8.07	14.67	12.85	9.15	8.57
Transformer	24.41	15.19	12.26	8.57	8.31	14.56	12.25	8.75	8.24
SBT	26.94	13.72	11.50	6.37	5.85	15.51	13.56	8.49	7.88
Tree-LSTM	26.61	21.63	17.69	14.27	14.15	21.04	17.48	15.86	15.79
Tree-COS	**29.17**	23.00	18.67	13.44	13.09	23.30	21.08	17.45	16.99
AST-AttendGRU	15.77	15.77	15.77	15.77	15.77	15.77	15.77	15.77	15.77
ECOS	23.91	23.91	23.91	23.91	23.91	23.91	23.91	23.91	23.91
VECOS	26.28	**26.28**	**26.28**	**26.28**	**26.28**	**26.28**	**26.28**	**26.28**	**26.28**

Table 1: Average BLEU scores of different models with different replacement approaches and ratios. VECOS, ECOS, and AST-AttendGRU yield stable performance under different δ, since the replacement mechanism of AST-AttendGRU and the erosion process of VECOS and ECOS replace identifiers with pre-defined symbols aforehand.

3.2 Experimental Settings

We use average BLEU [Papineni *et al.*, 2002] and Ribes [Isozaki *et al.*, 2010] as evaluation metrics to evaluate our approach measuring the correspondence between a generated summary and several output references. BLEU calculates the average precision over the n-gram mechanism with a penalty for short sequences, whose value varies from 0 to 100 as a percentage. Ribes takes into account the rank correlation coefficients with word precision, varying from 0 to 1. Higher BLEU and Ribes scores indicate better evaluation performance.

We choose eight approaches as baselines:

- **CodeNN**, **AttnGRU**, and **Transformer** [Iyer *et al.*, 2016; Cho *et al.*, 2014; Vaswani *et al.*, 2017]: Sequential neural approaches which treat code as plain text. CodeNN is the first neural-network-based approach based on LSTM for source code summarization. AttnGRU is a bi-directional GRU model with an attention mechanism. Transformer is a famous self-attention model proposed for neural machine translation.
- **SBT** [Hu *et al.*, 2018]: SBT is short for Structure-Based Traversal, a new way of converting an abstract syntax tree into a node sequence. An attention-based multi-layer LSTM processes the SBT node sequences to generate code summaries.
- **AST-AttendGRU** [LeClair *et al.*, 2019]: A structural GRU-based model specially designed for code with zero internal documentation. It replaces identifier names with a single '`<OTHER>`' token and takes the SBT sequence of the replaced AST as input.
- **Tree-LSTM** [Tai *et al.*, 2015]: A generalization of LSTM to tree-structured network topologies, which is used as the encoder to process AST. The decoder is vanilla attentional LSTM.
- **Tree-COS**, **ECOS**: Two variants of VECOS. Tree-COS refers to the auxiliary model ϕ described in Section 2.3 without erosion part τ. ECOS shares the same model structure with VECOS but is trained without variational semantics alignment.

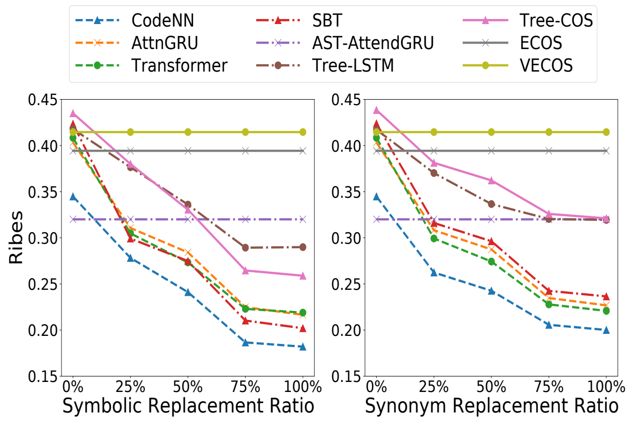


Figure 4: Ribes scores for different replacement ratios and approaches. VECOS and its variants are represented as solid lines. Other baselines are represented as dashed lines.

For our generative summarization model ψ of VECOS, both the input size and hidden size of LSTM are set as 256. The size of the latent variable z is 100. Both the internal LSTM of the encoder and the sequential LSTM for the decoder has only one layer. The initial learning rate is 0.01, and it will decay on learning plateaus with a decaying speed of 0.5. We take the sigmoid annealing strategy to adjust the coefficient λ_{kl} dynamically from 1.0 to 0.01 [Bowman *et al.*, 2015]. The weight decay factor for L2-regularization is set as 0.00001. The batch size for the training set is set as 100. The training process will stop when the performance of the validation set stops increasing for three epochs. For a fair comparison, the embedding size and hidden size of baselines are all set as 256. All the experiments are conducted with a machine equipped with one Intel Xeon E5-2620 CPU, 32GB RAM, a Nvidia Titan X graphic card with 12GB graphic storage. The operating system is Ubuntu 16.04.

3.3 Results and Analysis

Let δ denotes the replacement ratio. Table 1 and Figure 4 show the BLEU and Ribes scores of different models on test data with different δ and replacement approaches. From the statistics, VECOS significantly outperforms other approaches for various lexically confusing circumstances ($\delta > 0$). When code is lexically precise (δ=0), VECOS's performance is still comparable with previous approaches. The following is a detailed comparative and ablation analysis.

Comparative Analysis

Without any replacement (δ=0), the auxiliary model (Tree-COS) achieves the best performance. Compared with Tree-LSTM, Tree-COS improves 2.65 BLEU points, indicating the effectiveness of introducing latent variable z to represent functional semantics explicitly. Other approaches that considering structural information (Tree-LSTM, SBT) performs better than approaches that treat code as plain text (CodeNN, AttnGRU, Transformer). Although equipped with the erosion part, VECOS can still achieve comparable performance with most of the baselines for lexically precise code.

When we replace parts of the precisely defined identifiers ($\delta > 0$), the performance of sequential approaches (CodeNN, AttnGRU, Transformer) drops sharply. Even 25% replacement will result in half of the performance loss. When δ reaches 100%, their BLEU scores go below 10. Although SBT considers structural information by taking code's flatten AST as input, it still performs poorly when $\delta > 0$. We think this is because the sequential representation of AST is susceptible to disturbance. It is hard for neural models to adapt to such unseen data disturbance. Approaches that retain real tree-structured network topologies have better generalization performance under various lexically confusing circumstances, like Tree-LSTM and Tree-COS. However, they still suffer a decline when δ rises.

VECOS, ECOS, and AST-AttendGRU have a stable performance for different δ. This is because the replacement mechanism of AST-AttendGRU and the erosion process τ of VECOS and ECOS replace identifier names with pre-defined symbols aforehand. As δ increases, those approaches' advantage over other baselines gets more prominent. For the evaluation results, VECOS yields much better performance than AST-AttenGRU for both BLEU and Ribes. We think there are two main reasons. Firstly, AST-AttendGRU replaces all the identifiers with a single '`<OTHER>`' token, which changes the original code's functional semantics. Secondly, AST-AttendGRU attempts to learn the mapping from the replaced code to summaries directly, which is quite challenging because of the lack of lexical information and the changed functionality.

Ablation Analysis

We compare the performance of Tree-COS, ECOS, and VECOS to verify the effectiveness of each component of our approach. Taking away the erosion process τ, Tree-COS can achieve the highest BLEU score of lexically precise code ($\delta = 0$) but suffers decline when δ goes up. Equipped with the erosion part τ but without variational alignment training, ECOS can achieve a stable BLEU score of 23.91. With the variational alignment training, VECOS can improve the BLEU score to 26.28, which is a 10 percent increase over ECOS. In summary, the combination of erosion process and variational alignment ensures VECOS's excellent generalization towards various lexically confusing circumstances.

4 Related Work

Code Summarization

Automatic source code summarization is a hot topic in software engineering. Early studies in this field usually utilize techniques of text summarization or information retrieval for summary generation [Haiduc *et al.*, 2010]. Approaches like AutoComment [Wong *et al.*, 2013] try to leverage code-description mappings collect from StackOverflow and generate comments via code similarity. These approaches are usually evaluated by experienced programmers manually.

Since neural networks have achieved remarkable success in machine translation, researchers start to model this problem as a translation from programming language to natural language. CodeNN [Iyer *et al.*, 2016] and CodeAttention [Zheng *et al.*, 2019] utilize attention LSTM to generate summaries sequentially. To extract structural information, [Hu *et al.*, 2018] propose SBT as a new way to traverse abstract syntax tree so that sequential models can encode tree-structural data. Both AST-AttendGRU [LeClair *et al.*, 2019] and Hybrid-DRL [Wan *et al.*, 2018] try to encode structural and textual information independently and fuse for later comment decoding. Hybrid-DRL [Wan *et al.*, 2018] introduces reinforcement learning into this field for decoding to avoid exposure bias problem. AST-AttendGRU [LeClair *et al.*, 2019] proposes to generate code summaries without internal documentation.

Variational Neural Model

Variational AutoEncoder (VAE) is the first variational neural model proposed by [Kingma and Welling, 2013] for image generation, in which the prior of the latent variable is assigned as a standard Gaussian. The following works like [Zhang *et al.*, 2016; Serban *et al.*, 2017] extend it to a general variational encoder-decoder (VED) form for various NLP tasks like machine translation, sentence generation, and dialog systems. With the help of latent variables, those approaches usually yield better performances than vanilla attentional RNN models. On their basis, [Bahuleyan *et al.*, 2018] take a step forward by modeling attention mechanism via variational inferences to avoid bypassing phenomenon.

5 Conclusions

This paper proposes a novel framework called VECOS to generate reliable summaries for lexically confusing code. VECOS introduces an erosion process to map arbitrary source code to a unified eroded version to fit various real-world circumstances. With the help of variational alignment training, VECOS learns to extract functional semantics in a latent space $\mathbf{Z}$ by explicitly modeling the erosion process. The evaluation results show that VECOS is reliable to generate coherent summaries for various lexically confusing code. An interesting future work can be applying variational semantics alignment to more general scenarios such as code representation learning with data augmentation.

Acknowledgements

This research was supported by NSFC (62076121) and National Key Research and Development Program (2017YFB1001903).

References

[Bahuleyan *et al.*, 2018] Hareesh Bahuleyan, Lili Mou, Olga Vechtomova, and Pascal Poupart. Variational attention for sequence-to-sequence models. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1672–1682, 2018.

[Bavishi *et al.*, 2018] Rohan Bavishi, Michael Pradel, and Koushik Sen. Context2name: A deep learning-based approach to infer natural variable names from usage contexts. *arXiv preprint arXiv:1809.05193*, 2018.

[Bowman *et al.*, 2015] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[Cho *et al.*, 2014] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[Deissenboeck and Pizka, 2006] Florian Deissenboeck and Markus Pizka. Concise and consistent naming. *Software Quality Journal*, 14(3):261–282, 2006.

[Haiduc *et al.*, 2010] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. On the use of automated text summarization techniques for summarizing source code. In *Proceedings of the 17th Working Conference on Reverse Engineering*, pages 35–44. IEEE, 2010.

[Hu *et al.*, 2018] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension*, pages 200–210. IEEE, 2018.

[Husain *et al.*, 2019] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[Isozaki *et al.*, 2010] Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 14th Conference on Empirical Methods in Natural Language Processing*, pages 944–952, 2010.

[Iyer *et al.*, 2016] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the 54th annual meeting of the Association for Computational Linguistics*, pages 2073–2083, 2016.

[Kingma and Welling, 2013] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[LeClair *et al.*, 2019] Alexander LeClair, Siyuan Jiang, and Collin McMillan. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the 41st International Conference on Software Engineering*, pages 795–806. IEEE, 2019.

[Moreno *et al.*, 2013] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker. Automatic generation of natural language summaries for java classes. In *Proceedings of the 21st IEEE/ACM International Conference on Program Comprehension*, pages 23–32. IEEE, 2013.

[Papineni *et al.*, 2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[Pennington *et al.*, 2014] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 19th Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[Serban *et al.*, 2017] Iulian Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[Tai *et al.*, 2015] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[Tran *et al.*, 2019] Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N Nguyen. Recovering variable names for minified code with usage contexts. In *Proceedings of 41st International Conference on Software Engineering*, pages 1165–1175. IEEE, 2019.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. 2017.

[Wan *et al.*, 2018] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE international conference on Automated Software Engineering*, pages 397–407. ACM, 2018.

[Wong *et al.*, 2013] E. Wong, Jinqiu Yang, and Lin Tan. Autocomment: Mining question and answer sites for automatic comment generation. In *Proceedings of the 28th ACM/IEEE international conference on Automated Software Engineering*, pages 562–567. ACM, 2013.

[Zhang *et al.*, 2016] Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. Variational neural machine translation. In *Proceedings of the 21st Conference on Empirical Methods in Natural Language Processing*, pages 521–530, 2016.

[Zheng *et al.*, 2019] Wenhao Zheng, Hongyu Zhou, Ming Li, and Jianxin Wu. Codeattention: translating source code to comments by exploiting the code constructs. *Frontiers of Computer Science*, 13(3):565–578, 2019.