# Tram: A Token-level Retrieval-augmented Mechanism for Source Code Summarization

**Anonymous ACL submission**


## Abstract

Automatically generating human-readable text describing the functionality of a program is the intent of source code summarization. Although Neural Language Models achieve significant performance in this field, an emerging trend is combining neural models with external knowledge. Most previous approaches rely on the sentence-level retrieval and combination paradigm (retrieval of similar code snippets and use of the corresponding code and summary pairs) on the encoder side. However, this paradigm is coarse-grained and cannot directly take advantage of the high-quality retrieved summary tokens on the decoder side. In this paper, we explore a fine-grained token-level retrieval-augmented mechanism on the decoder side to help the vanilla neural model generate a better code summary. Furthermore, to mitigate the limitation of token-level retrieval on capturing contextual code semantics, we propose to integrate code semantics representation into summary tokens. Extensive experiments and human evaluation reveal that our token-level retrieval-augmented approach significantly improves performance and is more interpretive. We have made our code publicly available[1] to facilitate future research.


## 1 Introduction

With software functions becoming more comprehensive and complex, it becomes a heavy burden for developers to understand software. It has been reported that nearly 90% (Wan et al., 2018) of effort is used for maintenance, and much of this effort is spent on understanding the maintenance task and related software source codes. Source code summary as a natural language is indispensable in software, since humans can easily read and understand it, as shown in Table 1. However, manually writing source code summaries is time-consuming and tedious. Besides, in the process of continuous software iteration, the source code summary is often outdated. Hence, automatically generating concise and human-readable source code summaries is critical and meaningful.

```python
def cos(x):
   np = import module("numpy")
   if isinstance(x, (int, float)):
      return interval(np.sin(x))
   elif isinstance(x, interval):
      if (not(np.isifnite(x.start) and
            np.isfinite(x.end))):
         return interval((-1), 1, is_valid=x.is_valid)
      (na, _) = divmod(x.start, (np.pi / 2.0))
      (nb, _) = divmod(x.end, (np.pi / 2.0))
      start = min(np.cos(x.start), np.cos(x.end))
      end = max(np.cos(x.start), np.cos(x.end))
      if ((nb - na) > 4):
         return interval((-1), 1, is_valid=x.is_valid)
      elif (na == nb):
         return interval(start, end, is_valid=x.is_valid)
      else:
         if ((na // 4) != (nb // 4)):
            end = 1
         if (((na - 2) // 4) != ((nb - 2) // 4)):
            start = -1
         return interval(start, end, is_valid=x.is_valid)
   else:
      raise NotImplementedError
```

Summary: evaluates the **cos** of an interval.

*Sentence-level:*

evaluates logarithm to base 10 of an interval.

*Token-level:* cos, tangent, sin, hyperbolic $\cdots$

Table 1: Task sample of source code summarization. The example is a Python function instance.

With the development of language models and the linguistic nature of source code, researchers explored Seq2Seq architecture such as recurrent neural networks to generate summaries from the given source code (Iyer et al., 2016; Loyola et al., 2017; Liang and Zhu, 2018). Soon afterward, Transformer-based models (Ahmad et al., 2020; Wu et al., 2021; Gong et al., 2022) were proposed, which outperformed previous RNN-based models by a large margin. Recently, many approaches propose to additionally exploit the structural properties of source code, including Abstract Syntax Tree (AST), Program Dependency Graph (PDG), etc. Current structure-aware methods fuse structural information in hybrid way (Hu et al., 2018; Shido

[1] https://anonymous.4open.science/r/SourceCodeSummary-8ABD

et al., 2019; LeClair et al., 2020; Choi et al., 2021; Shi et al., 2021), or structured-guided way (Wu et al., 2021; Son et al., 2022; Gong et al., 2022). While these methods achieve excellent results, they only focus on mining the information of the code itself to get richer code representation, neglecting the existing human-written code-summary pairs.

In order to make use of the external existing high-quality code and the corresponding summary instances, Liu et al. (2021) retrieved the most similar code snippet by text similarity metric to enrich target code structure information for getting a better code representation encoder. This retrieval method only carries out from the perspective of text similarity and neglects code semantic similarity in the retrieval phase. Besides, the summary corresponding to the retrieved code snippet is just a simple concatenate to the encoder. Zhang et al. (2020); Parvez et al. (2021) used a pre-trained encoder to obtain code semantic representation, which was used to retrieve similar code snippets. The former only used similar code snippets and discarded the corresponding summaries; the latter directly spliced the retrieved code snippet and the corresponding summary behind the target code; both were also aimed at better code representation on the encoder side. Code summarization, as a generative task essentially, the decoder generates the summary tokens autoregressively. However, previous retrieval-augmented methods neglect to fuse the retrieved information on the decoder side, which will result in the utilization pattern being indirect and insufficient. Besides, current retrieval-augmented methods that use the summary are still at the coarse-grained sentence level (i.e., concatenate), which will blend in a lot of noise, as shown in Table 1, many of the corresponding summary tokens are not related, like *"logarithm to base 10"*.

This inspires us to perform a fine-grained retrieval manner on the decoder side, so we propose a token-level retrieval-augmented mechanism. In order to achieve the purpose of retrieving semantic similar summary tokens, we first construct a datastore to store the summary token and corresponding token representation through a pre-trained base model offline. At the same time, in order to fully consider contextual code semantics associated with summary tokens, our token representation integrates code representation with attention weight. The summary token representation at each generation step is used to retrieve the most similar top-$K$ tokens, as shown in Table 1, the token-level retrieval results are *"cos, tangent, sin, hyperbolic $\cdots$"* at the generation step of next token *"cos"*. The retrieved top-$K$ tokens are expanded to a probability distribution called retrieval-based distribution. The retrieval-based distribution fused with the vanilla distribution to form the final distribution. Besides, our token-level retrieval mechanism can be seamlessly integrated with the additional sentence-level retrieval manner.

In summary, the main contributions of this paper are outlined as follows:

1. We first explore a token-level retrieval-augmented mechanism on the decoder side for source code summarization.

2. Our proposed retrieval-augmented mechanism is orthogonal to existing improvements, e.g. combined with code representation or addition sentence-level retrieval manner.

3. Extensive experiments and human evaluation show that our proposed method significantly outperforms other baseline models.

## 2 Methodology

In this work, we propose a Token-level Retrieval-augmented Mechanism for Source Code Summarization (Tram). Firstly, we introduce the base model, which establishes the summary token and corresponding token representation pairs in a datastore. Then, we formulate the token-level retrieval method, which retrieves tokens from the datastore and blends the final prediction probability. Finally, we introduce the additional sentence-level similar code snippet retrieval-augmented manner. The overview of Tram is shown in Figure 1.

### 2.1 Base Model

In the first place, we use Transformer (Vaswani et al., 2017) as our backbone. The Transformer consists of stacked multi-head attention and parameterized linear transformation layers for both encoder and decoder. Each layer emphasizes on self-attention mechanism, which is denoted as:

$$e_{ij} = \frac{x_i W^Q (x_j W^K)^T}{\sqrt{d_k}}$$

$$h_i = \sum_{j=1}^{n} \alpha_{ij} (w_j W^V)$$

where $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j=1}^{n} \exp(e_{ij})}$, $W^Q, W^K, W^V$ are the parameters that are unique per layer and attention

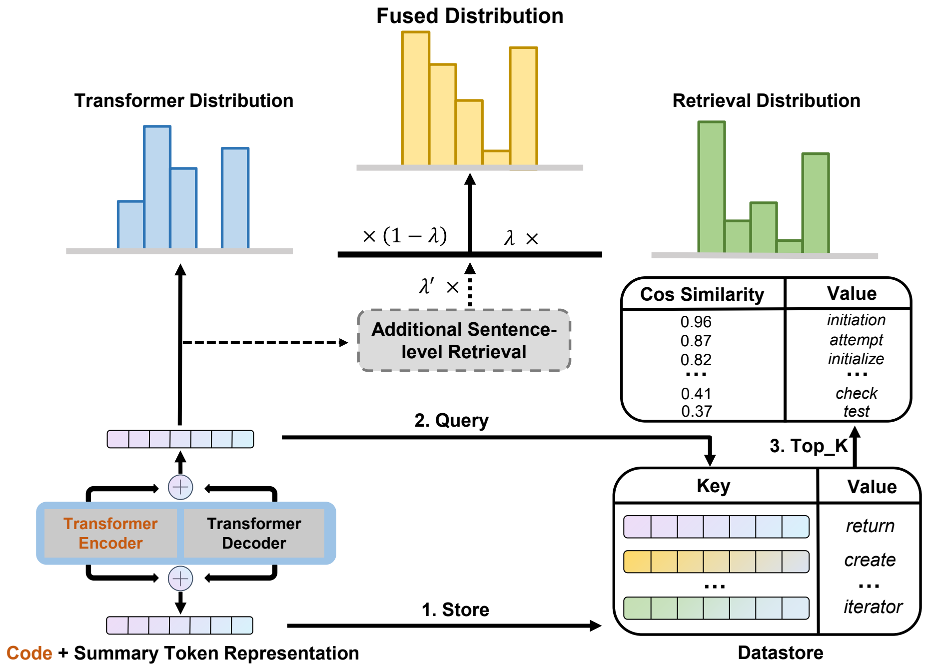


Figure 1: The overview architecture of Trams. The left part is the base model and the vanilla model distribution. The right part shows the process of retrieving similar summary tokens and getting the retrieval-based distribution. The Query and Key are the combinations of encoder and decoder representation.

head, $h_i$ is the $i$th-token hidden representation.

Nevertheless, as pointed out in Ahmad et al. (2020), the semantic representation of a code does not rely on the absolute positions of its tokens. Instead, their mutual interactions influence the meaning of the source code. To encode the pairwise relationships between input elements, Shaw et al. (2018) extend the self-attention mechanism as follows:

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_k}}$$

$$h_i = \sum_{j=1}^{n} \alpha_{ij} (w_j W^V + a_{ij}^V)$$

where $a_{ij}^K$ and $a_{ij}^V$ are relative positional representation for the two position $i$ and $j$. We clip the maximum relative position to a maximum absolute value of $l$ because precise relative position information is not useful beyond a certain distance.

$$a_{ij}^K = \omega_{clip(j-i,l)}^K, a_{ij}^V = \omega_{clip(j-i,l)}^V$$

$$clip(x, l) = max(-l, min(x, l))$$

Hence, the Transformer architecture equipped with relative position representation serves as our base model.

## 2.2 Datastore Creation

For fine-grained token-level retrieval, the datastore that store summary token representation and corresponding token pairs is indispensable. At the stage of datastore establishment, we adopt the above pre-trained base model to go through all training instances $(C, S)$ in an offline manner. The encoder encodes the source code into a sequence of hidden states. The decoder takes the representations of the source code as input and generates target summary text autoregressively. During this process, for each instance $(c, s)$, we record encoder representation (which contains code semantic) as $CR$, decoder presentation[2] (which contain summary semantic) as $SR$ and corresponding ground-truth target token as $s$. The representation and target token are stored as key and value, respectively. Formally, given a training set, we construct the datastore as follows:

$$(\mathcal{K}, \mathcal{V}) = \{(R_t, s_t), \forall s_t \in s | (c, s) \in (C, S)\}$$

where the value $s_t$ is the ground-truth target token with $t$ denoting decoding timestep, $R_t$ is the corresponding hidden representation. Spread it out, $R_t$ consist of two parts: one is code representation

[2] It is worth noting that we record the hidden representation input to the final layer feed network in the decoder.

$CR_t$, the other is summary token representation $SR_t$:

$$R_t = Trans(CR_t, SR_t)$$

$Trans(\cdot)$ can be any aggregation transformation. In this work, we use the *concatenate* operation. Especially, $CR_t$ is the weighted sum of each code token representation.

$$CR_t = \sum_{i=1}^{\ell} \widetilde{w}_t \cdot cr_i$$

where $\widetilde{w}_t$ is the cross attention weight, $cr_i$ is the $i$-th code token representation and $\ell$ denoted code token length.

### 2.3 Token-level Retrieval

While inference, at each decoding step $t$, the decoder representation $SR_t$ together with code representation $CR_t$ used the same transformation operator $Trans(\cdot)$ as query $q_t$. The query then retrieves the top-$K$ most similar summary tokens in the datastore according to cos similarity distance. It is worth noting that we use cos similarity instead of squared-$L^2$ distance because of the performance of the preliminary experiment. As an added bonus, cos similarity can be seen as retrieval confidence. In practice, the retrieval over millions of key-value pairs is carried out using FAISS (Johnson et al., 2019), a library for fast nearest neighbor search in high-dimensional spaces. The retrieved key-value pairs $(k, v)$ and corresponding cos similarity distance $\alpha$ composed a triple set $\mathcal{N} = \{(k_i, v_i, \alpha_i) | i = 1, 2, \cdots, K\}$. Inspired by KNN-MT (Khandelwal et al., 2021), the triple set can then be expanded and normalized to the retrieval-based distribution as follows:

$$P_r(s_t|c, \hat{s}_{<t}) \propto \sum_{(k_i, v_i, \alpha_i) \in \mathcal{N}} \mathbb{1}_{s_t = v_i} \exp\left(g(k_i, \alpha_i)\right)$$

$$g(k_i, \alpha_i) = \alpha_i * T$$

where $g(\cdot)$ can be any Kernel Density Estimation (KDE), in our paper, we use the product form; $T$ is the temperature to regulate probability distribution.

### 2.4 Fused Distribution

The final prediction distribution can be seen as the vanilla base model output distribution and the retrieval-based distribution are interpolated by a hyper-parameter $\lambda$:

$$\begin{aligned} P(s_t|c, \hat{s}_{<t}) &= \lambda * P_r(s_t|c, \hat{s}_{<t}) \\ &+ (1 - \lambda) * P_m(s_t|c, \hat{s}_{<t}) \end{aligned}$$

| Datasets | Java | Python | CCSD |
|---|---|---|---|
| Train | 69,708 | 55,538 | 84,316 |
| Validation | 8,714 | 18,505 | 4,432 |
| Test | 8,714 | 18,502 | 4,203 |
| Code: Avg. tokens | 73.76 | 49.42 | 68.59 |
| Summary: Avg. tokens | 17.73 | 9.48 | 8.45 |

Table 2: Statistics of the experimental datasets. We split CCSD following Liu et al. (2021), and the Java/Python dataset splits are public available.

where $P_m$ indicates the vanilla base model distribution.

### 2.5 Additional Sentence-level Retrieval

The token-level retrieval-augmented method can also be seamlessly incorporated with additional sentence-level retrieval. Additional sentence-level retrieval means finding the most semantic similarity code snippet, and using an additional encoder to encode the code snippet, then decoding with the target code snippet synchronously. Formally, the final fused distribution can be extended as follow:

$$\begin{aligned} P(s_t|c, \hat{s}_{<t}) &= \lambda_1 * P_r(s_t|c, \hat{s}_{<t}) \\ &+ \lambda_2 * Sim * P_s(s_t|\langle c \rangle, \hat{s}_{<t}) \\ &+ (1 - \lambda_1 - \lambda_2) * P_m(s_t|c, \hat{s}_{<t}) \end{aligned}$$

where $P_s$ is the additional sentence-level produced distribution, $\langle c \rangle$ is the most semantic similar code snippet to $c$, and $Sim$ is the corresponding similarity score.

## 3 Experiments

### 3.1 Experimental Setup

**Datasets.** We conduct the source code summarization experiments on three public benchmarks of Java (Hu et al., 2018), Python (Wan et al., 2018), CCSD (C Code Summarization Dataset) (Liu et al., 2021). The partitioning of train/validation/test sets follows the original datasets. The statistics of the three datasets are shown in Table 2.

**Out-of-Vocabulary.** The vast operators and identifiers in program language may produce a much larger vocabulary than natural language, which can cause Out-of-Vocabulary problem. To avoid this problem, we apply $CamelCase$ and $snake\_case$ tokenizers that are consistent with recent works (Gong et al., 2022; Wu et al., 2021; Ahmad et al., 2020) to reduce the vocabulary size of source code.

**Metrics.** Similar to recent work (Gong et al., 2022; Son et al., 2022), we evaluate the source code summarization performance using three widely-used metrics, BLEU (Papineni et al., 2002), METEOR (Banerjee and Lavie, 2005) and ROUGE-L (Lin, 2004). These metrics are prevalent in machine translation and text summarization. Furthermore, considering the essence of source code summarization to help humans better understand code, we also conduct a human evaluation study. The volunteers are asked to rank summaries generated from the anonymized approaches from 1 to 5 (i.e., 1: Poor, 2: Marginal, 3: Acceptable, 4: Good, 5: Excellent) based on *Similarity*, *Relevance*, and *Fluency*. Further details on human evaluation can be found in Appendix A.

**Training Details.** We implement our approach based on JoeyNMT (Kreutzer et al., 2019) on NVIDIA 3090. The batch size is set to 32 and Adam optimizer is used with an initial learning rate $10^{-4}$. To alleviate overfitting, we adopt early stopping with patience 15. For Faiss (Johnson et al., 2019) Index, we employ IndexFlatIP and top-$K$=16 to keep a balance between retrieval quality and retrieval speed in the large scale datastore. It is worth noting that the only part that requires training is the base model, and once trained, all parameters of the base model are fixed.

### 3.2 Baselines

**RNN-based.** CODE-NN (Iyer et al., 2016) follows LSTM-based encoder-decoder architecture with attention mechanism, treating source code as natural language. Tree2Seq (Eriguchi et al., 2016) is an end-to-end syntactic NMT model which directly uses a tree-based LSTM as an encoder. It extends an RNN model with the source code structure. Hybrid2Seq (Wan et al., 2018) incorporates ASTs and sequential content of code snippets into a deep reinforcement learning framework. DeepCom (Hu et al., 2018) flattens the AST into a sequence as input, which is obtained via traversing the AST with a structure-based traversal (SBT) method. Dual Model (Wei et al., 2019) treats code summarization and code generation as a dual task. It trains the two tasks jointly by a dual training framework to simultaneously improve the performance of both tasks.

**Transformer-based.** Transformer (Ahmad et al., 2020) is the first attempt to use transformer architecture, equipped with relative positional encoding and copy mechanism (See et al., 2017), effectively capturing long-range dependencies of source code. CAST (Shi et al., 2021) hierarchically splits a large AST into a set of subtrees and utilizes a recursive neural network to encode the subtrees, aimed to capture the rich information in ASTs. mAST + GCN (Choi et al., 2021) adopt the AST and graph convolution to model the structural information and the Transformer to model the sequential information. SiT (Wu et al., 2021) incorporates a multi-view graph matrix into Transformer's self-attention mechanism. Essentially, it improves performance by masking redundant attention in the calculation process of self-attention scores. SiT + PDG (Son et al., 2022) pointed program dependency graph is more effective for expressing the structural information than AST. SCRIPT (Gong et al., 2022) utilizes AST structural relative positions to augment the structural correlations between code tokens.

**Retrieval-based.** Rencos (Zhang et al., 2020) is the first retrieval-based Seq2Seq model, which computes a joint probability conditioned on both original source code and retrieved the most similar source code for a summary generation. HGNN (Liu et al., 2021) is the retrieval-based GNN model, which retrieval the most similar code and uses a Hybrid GNN by fusing static graph and dynamic graph to capture global code graph information.

### 3.3 Main Results

The main experiment results are shown in Table 3 and Table 4 in terms of the three automatic evaluation metrics. The reason for having two tables is most recently Transformer-based works compared their performance on the two widely used Java and Python benchmarks; the recently Retrieval-based works compared on different benchmarks. Thus, our experiments are performed on all three datasets (Java/Python/CCSD) for more comprehensive comparison. We calculate the values of the metrics following the same scripts. For these metrics, the larger value indicates better performance.

From Table 3, SiT + PDG and SCRIPT outperform all previous works by a significant margin. However, our proposed token-level retrieval-augmented model further boosts results with 1.25 BLEU points on Java and 1.74 BLEU points on Python and achieves new state-of-the-art results. At the same time, we notice that the performance improvement of Python is better than that of Java. The main reason we speculate is that the average

| Model | Java | | | Python | | |
|---|---|---|---|---|---|---|
| | **BLEU** | **ROUGE-L** | **METEOR** | **BLEU** | **ROUGE-L** | **METEOR** |
| *RNN-based Methods* | | | | | | |
| CODE-NN (Iyer et al., 2016) | 27.60 | 41.10 | 12.61 | 17.36 | 37.81 | 09.29 |
| Tree2Seq (Eriguchi et al., 2016) | 37.88 | 51.50 | 22.55 | 20.07 | 35.64 | 08.96 |
| Hybrid2Seq (Wan et al., 2018) | 38.22 | 51.91 | 22.75 | 19.28 | 39.34 | 09.75 |
| DeepCom (Hu et al., 2018) | 39.75 | 52.67 | 23.06 | 20.78 | 37.35 | 09.98 |
| Dual Model (Wei et al., 2019) | 42.39 | 53.61 | 25.77 | 21.80 | 39.45 | 11.14 |
| *Transformer-based Methods* | | | | | | |
| Transformer (Ahmad et al., 2020) | 44.58 | 54.76 | 26.43 | 32.52 | 46.73 | 19.77 |
| CAST (Shi et al., 2021) | 45.19 | 55.08 | 27.88 | - | - | - |
| mAST + GCN (Choi et al., 2021) | 45.49 | 54.82 | 27.17 | 32.82 | 46.81 | 20.12 |
| SiT (Wu et al., 2021) | 45.70 | 55.54 | 27.55 | 33.46 | 47.50 | 20.28 |
| SiT + PDG (Son et al., 2022) | 46.86 | 56.69 | - | - | - | - |
| SCRIPT (Gong et al., 2022) | 46.89 | 56.69 | 28.48 | 34.00 | 48.15 | 20.84 |
| *Our Method* | | | | | | |
| Base | 46.72 | 56.74 | 28.60 | 34.01 | 48.21 | 20.93 |
| Tram | **48.14** | **57.89** | **29.38** | **35.74** | **49.63** | **21.87** |
| Tram w/o CR | 47.96 | 57.42 | 29.23 | 35.51 | 49.37 | 21.68 |
| *Tram with SenRe* | *48.37* | *58.21* | *29.69* | *36.15* | *49.76* | *22.03* |

Table 3: Comparison of the performance of our method with other baseline methods on Java and Python benchmarks in terms of BLEU, ROUGE-L, and METEOR. The results of base models are reported in their original papers. '-' refers to no corresponding value from the paper. CR refers to code representation, SenRe refers to additional sentence-level retrieval. All of our methods are the mean of 5 runs with different random seeds.

code token length of Java is longer (from Table 2) and has richer code structure information, and the Transformer-based structure-induced methods can capture richer code semantics in their customized encoder.

Table 4 compares our proposed model with other retrieval-based models on CCSD and Python benchmarks. Our base model is even comparable to other retrieval-based methods; the main reason is that the backbone [3] are different. We reproduce Rencos architecture[4] in our base model for fair comparison, which we denoted as $Base + Rencos$. Our model still outperforms other retrieval-based methods, further improving performance with 2.05 BLEU points and 1.47 BLEU points on CCSD and Python, respectively. This also proves the superiority of our fine-grained retrieval-augmented method to fuse similar summary tokens on the decoder side.

### 3.4 Ablation Study

To validate the effectiveness of Code Representation (CR), we eliminate CR for comparison. The performance decline in all datasets demonstrated that the fusion with code semantic representation into the summary token is also important for summary token retrieval.

As pointed out in the methodology, our retrieval-augmented method can also be seamlessly incorporated with additional sentence-level retrieval (*Tram with SenRe*). The results show *Tram with SenRe* improved by 0.23 BLEU, 0.41 BLEU, and 0.75 BLEU points on Java, Python, and CCSD, respectively. The performance improvement of *Tram with SenRe* demonstrated the superiority of the combination of sentence-level retrieval manner and token-level retrieval manner, the formal aimed at retrieving the most similar code snippet and fused on the encoder side, and the latter aimed at retrieving the most similar summary token and fused on the decoder side; both are beneficial.

## 4 Analysis

### 4.1 Hyper-parameters Analysis

Our methods have two main hyper-parameters: $\lambda$ and $T$. $\lambda$ means the weight of the retrieval-based distribution part to account for the final distribution; the bigger value means the final distribution relies more on retrieval results and vice versa. $T$ means Temperature, which smooths the retrieval-based distribution. We plot the performance of Tram with different hyper-parameter selections in Figure

[3]other retrieval-based methods are RNN-based.

[4]HGNN code is not open source.

| Model | CCSD | | | Python‡ | | |
|---|---|---|---|---|---|---|
| | **BLEU** | **ROUGE-L** | **METEOR** | **BLEU** | **ROUGE-L** | **METEOR** |
| *Retrieval-based Methods* | | | | | | |
| Rencos (Zhang et al., 2020) | 14.80 | 31.41 | 14.64 | 34.73 | 47.53 | 21.06 |
| HGNN (Liu et al., 2021) | 16.72 | 34.29 | 16.25 | - | - | - |
| *Our Method* | | | | | | |
| Base | 17.82 | 35.33 | 16.71 | 34.85 | 48.84 | 21.49 |
| Base + Rencos | 19.43 | 36.92 | 17.69 | 35.26 | 49.25 | 22.07 |
| Tram | **21.48** | **37.88** | **18.35** | **36.73** | **50.35** | **22.53** |
| Tram w/o CR | 21.30 | 37.77 | 18.22 | 36.53 | 50.20 | 22.35 |
| *Tram with SenRe* | *22.23* | *38.16* | *18.96* | *36.95* | *50.69* | *22.93* |

Table 4: Comparison of ours with other retrieval methods. CR means code representation, SenRe means additional sentence-level retrieval. ‡ means the Python dataset is slightly different from the Python on Tabel 3, and we are consistent with Rencos (Zhang et al., 2020). All of our methods are the mean of 5 runs with different random seeds.

| Model | Java | | | Python‡ | | |
|---|---|---|---|---|---|---|
| | **Similarity** | **Relevance** | **Fluency** | **Similarity** | **Relevance** | **Fluency** |
| Rencos | - | - | - | 3.07 | 3.06 | 3.96 |
| SCRIPT | 3.65 | 3.70 | 4.12 | - | - | - |
| Base | 3.62 | 3.64 | 4.10 | 3.20 | 3.24 | 4.03 |
| Tram | **3.83** | **3.89** | **4.23** | **3.33** | **3.44** | **4.14** |

Table 5: Human Evaluation on Java and Python datasets.

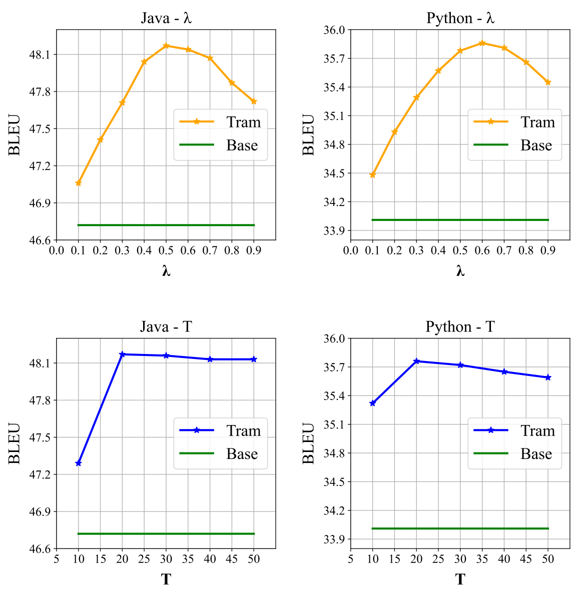


Figure 2: The study of hyper-parameters ($\lambda$ and $T$) selections in Java and Python datasets.

2. For $\lambda$, we find different $\lambda$ selections have a significant impact on the final performance, and for different datasets, the optimal $\lambda$ is different (i.e., $\lambda = 0.5$ for Java and $\lambda = 0.6$ for Python). In addition, another interesting phenomenon is that all different $\lambda$ have a positive effect on the final result. For $T$, on the one hand, too small cannot separate the retrieval-based distribution; on the other hand, too large will cause the retrieval-based distribution to focus on only one token. The final result shows both declines the performance.

### 4.2 Human Evaluation

We perform a human evaluation to assess the quality of the generated summaries by our approach, Rencos, SCRIPT, and Base model in terms of *Similarity*, *Relevance*, and *Fluency* as shown in Table 5. The results show that our approach can generate better summaries that are more similar to the ground truth, more relevant to the source code, and more fluency in naturalness.

### 4.3 Qualitative Analysis

We provide a couple of examples in Table 6 to demonstrate the usefulness of our proposed approach qualitatively. The qualitative analysis reveals that, compared to other models, the token-level retrieval-augmented manner enables visualization of the *Retrieval Results* and corresponding probability at each generation step, as shown in the last line of each function instance, which makes our model better interpretability.

```c
void scsi_netlink_init(void){
   struct netlink_kernle_cfg cfg;
   cfg.input = scsi_nl_rcv_msg;
   cfg.groups = SCSI_NL_GPRP_CNT;
   scsi_nl_sock = netlink_kernel_create(&init_net,
   NETLINK_SCSITRANSPORT, &cfg);
   if (!scsi_nl_sock){
      printk(KERN_ERR "%s: register of receive handler failed\n", __func__);
      return;}
   return;}
```

Base: called by scsi netlink initialization to register the scsi netlink interface.
Rencos: called by scsi netlink interface to register the scsi netlink interface.
Tram: called by scsi **subsystem** to register the scsi transport netlink interface.
Human Written: called by scsi **subsystem** to initialize the scsi transport netlink interface.
***Retrieval Results:*** "*subsystem*" (0.90), "*transport*"(0.04), *"stack"*(0.02), *"command"*(0.0034), *"device"*(0.0025) · · ·

```python
def category_structure(category, site):
   return {'description': category.title,
   'html_Url': ('%s://%s%s'%(PROTOCOL, site.domain,
            category.get_absolute_url())),
   'rss_Url': ('%s://%s%s'%(PROTOCOL, site.domain,
            reverse('zinnia:category_feed', args=[category.tree_path]))),
   'category_Id': category.pk ,
   'parent_Id': ((category.parent and category.parent.pk) or 0 ),
   'category_Description': category.description,
   'category_Name': category.title }
```

Base: updates the structure.
Rencos: a post structure.
Tram: a **category** structure.
Human Written: a **category** structure.
***Retrieval Results:*** "*category*"(0.43), "*tag*"(0.11), "*post*"(0.07), "*helper*"(0.06),"*version*"(0.06) · · ·

Table 6: Task samples. The first one is a C instance, the second one is a Python instance. The bold red font is the keyword of the generated summary. The ***Retrieval Results*** line is the visible retrieval results and corresponding probability after $softmax$ on the keyword generation step.

### 4.4 Inference Speed

A common concern about retrieval-based methods is that additional retrieval processes may slow the inference speed. We test the inference speed in CCSD and Python‡ datasets. The average inference time of Tram is 1.28 times of base model, which is only slightly slower but has a speed of 1.96x compared to $Base + Rencos$ model.

## 5 Related Work

**Source Code Summarization** Recent works (Gong et al., 2022; Son et al., 2022; Peng et al., 2021; Shi et al., 2021; Wu et al., 2021) on source code summarization pay more and more attention to code structural information, including AST, Control dependency, PDG, etc. These works mainly focus on how to capture and exploit the structural information of the code itself. Our work is orthogonal to theirs, aimed at how to better and fine-grained blend existing high-quality human-written code-summary pairs.

**K-Nearest-Neighbor Machine Translation** Recently, non-parametric methods have been successfully applied to neural machine translation (Khandelwal et al., 2021; Jiang et al., 2021; Zheng et al., 2021a,b). These approaches complement advanced NMT models with external memory to alleviate the performance degradation in domain adaption. Compared to these works, we have intelligently integrated code semantics in the retrieval process, and our token-level retrieval-augmented mechanism can be integrated with other sentence-level retrieval methods.

## 6 Conclusion

In this paper, we proposed a novel token-level retrieval-augmented mechanism for source code summarization. By a well-designed fine-grained retrieval pattern, our method can effectively incorporate external human-written code-summary pairs on the decoder side. The extensive experiments and human evaluation show that our approach has a significant performance improvement. However, the limitation of our retrieval-augmented method is heavily relying on high-quality code-summary pairs; exploring how to deal with noisy and low-resource scenarios will be our future direction.

## References


Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4998–5007, Online. Association for Computational Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

YunSeok Choi, JinYeong Bak, CheolWon Na, and Jee-Hyong Lee. 2021. Learning sequential and structural information for source code summarization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2842–2851, Online. Association for Computational Linguistics.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 823–833, Berlin, Germany. Association for Computational Linguistics.

Zi Gong, Cuiyun Gao, Yasheng Wang, Wenchao Gu, Yun Peng, and Zenglin Xu. 2022. Source code summarization with structural relative position guided transformer. *arXiv preprint arXiv:2202.06521*.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, page 200–210, New York, NY, USA. Association for Computing Machinery.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Qingnan Jiang, Mingxuan Wang, Jun Cao, Shanbo Cheng, Shujian Huang, and Lei Li. 2021. Learning kernel-smoothed machine translation with retrieved examples. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7280–7290, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *International Conference on Learning Representations*.

Julia Kreutzer, Jasmijn Bastings, and Stefan Riezler. 2019. Joey NMT: A minimalist NMT toolkit for novices. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 109–114, Hong Kong, China. Association for Computational Linguistics.

Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan. 2020. Improved code summarization via a graph neural network. In *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, page 184–195, New York, NY, USA. Association for Computing Machinery.

Yuding Liang and Kenny Zhu. 2018. Automatic generation of text descriptive comments for code blocks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. 2021. Retrieval-augmented generation for code summarization via hybrid {gnn}. In *International Conference on Learning Representations*.

Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 287–292, Vancouver, Canada. Association for Computational Linguistics.

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 on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Md Rizwan Parvez, Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2719–2734, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Han Peng, Ge Li, Wenhan Wang, Yunfei Zhao, and Zhi Jin. 2021. Integrating tree path in transformer for code representation. In *Advances in Neural Information Processing Systems*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021. CAST: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4053–4062, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yusuke Shido, Yasuaki Kobayashi, Akihiro Yamamoto, Atsushi Miyamoto, and Tadayuki Matsumura. 2019. Automatic source code summarization with extended tree-lstm. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Jikyoeng Son, Joonghyuk Hahn, HyeonTae Seo, and Yo-Sub Han. 2022. Boosting code summarization by embedding code structures. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 5966–5977, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, page 397–407, New York, NY, USA. Association for Computing Machinery.

Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Hongqiu Wu, Hai Zhao, and Min Zhang. 2021. Code summarization with structure-induced transformer. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1078–1090, Online. Association for Computational Linguistics.

Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1385–1397, New York, NY, USA. Association for Computing Machinery.

Xin Zheng, Zhirui Zhang, Junliang Guo, Shujian Huang, Boxing Chen, Weihua Luo, and Jiajun Chen. 2021a. Adaptive nearest neighbor machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 368–374, Online. Association for Computational Linguistics.

Xin Zheng, Zhirui Zhang, Shujian Huang, Boxing Chen, Jun Xie, Weihua Luo, and Jiajun Chen. 2021b. Non-parametric unsupervised domain adaptation for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4234–4241, Punta Cana, Dominican Republic. Association for Computational Linguistics.

## A Human Evaluation

In our human evaluation, We invite 3 PhD students and 5 master students as volunteers, who have at least 2-5 years software engineering experiences. We conducted a small-scale random dataset (i.e., 100 random Java samples and 100 random Python samples). The volunteers are asked to rank summaries generated from the anonymized approaches from 1 to 5 (i.e., 1: Poor, 2: Marginal, 3: Acceptable, 4: Good, 5: Excellent) based on the three following questions:

- **Similarity**: How similarity of generated summary and ground-truth?
- **Relevance**: Is the generated summary relevant to the source code?
- **Fluency**: Is this generated summary syntactically correct and fluency?

For each evaluation summary, the rating scale is from 1 to 5, where a higher score means better quality. Responses from all volunteers were collected and averaged.