Multi-Modal Attention Network Learning for Semantic Source Code Retrieval

Yao Wan[*†‡], Jingdong Shu[*†], Yulei Sui[‡], Guandong Xu[‡], Zhou Zhao[†], Jian Wu[†] and Philip S. Yu[†◇]
[†]College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[‡]School of Computer Science, University of Technology Sydney, Australia
[†]Department of Computer Science, University of Illinois at Chicago, Illinois, USA
[◇]Institute for Data Science, Tsinghua University, Beijing, China
[♯]State Key Laboratory of Cognitive Intelligence, iFLYTEK, Hefei, China
{wanyao, jdshu, zhaozhou, wujian2000}@zju.edu.cn, {yulei.sui, guandong.xu}@uts.edu.au, psyu@uic.edu

***Abstract*—Code retrieval techniques and tools have been playing a key role in facilitating software developers to retrieve existing code fragments from available open-source repositories given a user query (e.g., a short natural language text describing the functionality for retrieving a particular code snippet). Despite the existing efforts in improving the effectiveness of code retrieval, there are still two main issues hindering them from being used to accurately retrieve satisfiable code fragments from large-scale repositories when answering complicated queries. First, the existing approaches only consider shallow features of source code such as method names and code tokens, but ignoring structured features such as abstract syntax trees (ASTs) and control-flow graphs (CFGs) of source code, which contains rich and well-defined semantics of source code. Second, although the deep learning-based approach performs well on the representation of source code, it lacks the explainability, making it hard to interpret the retrieval results and almost impossible to understand which features of source code contribute more to the final results.**

To tackle the two aforementioned issues, this paper proposes MMAN, a novel Multi-Modal Attention Network for semantic source code retrieval. A comprehensive multi-modal representation is developed for representing unstructured and structured features of source code, with one LSTM for the sequential tokens of code, a Tree-LSTM for the AST of code and a GGNN (Gated Graph Neural Network) for the CFG of code. Furthermore, a multi-modal attention fusion layer is applied to assign weights to different parts of each modality of source code and then integrate them into a single hybrid representation. Comprehensive experiments and analysis on a large-scale real-world dataset show that our proposed model can accurately retrieve code snippets and outperforms the state-of-the-art methods.

***Index Terms*—Code retrieval, multi-modal network, attention mechanism, deep learning.**

I. Introduction

With the advent of immense source code repositories such as GitHub [1] and StackOverflow [2], it is gradually becoming a key software development activity for programmers to search existing code with the same functionality, and reuse as much of that code as possible [3]. The goal of code retrieval is to retrieve a particular code fragment from available open-source repositories given a user specification (e.g., a short text describing the functionality of the code fragment). The key challenges of implementing such a code retrieval system lie in two folds: (a) a deep semantic understanding of the source code and (b) measuring the similarity of cross modalities (i.e., input natural language and source code).

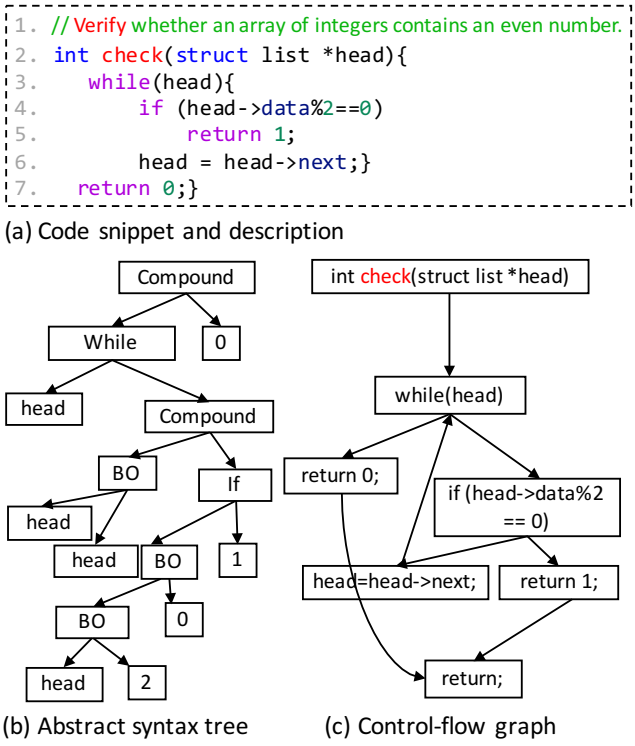


Figure 1: A motivating example to better illustrate our motivation. (a) A code snippet and its corresponding description. (b) The AST of the code snippet. (c) The control-flow graph of the code snippet.

Existing Efforts and Limitations. Many existing efforts have been made towards searching the huge amount of available code resources for a natural language query, ranging from keyword matching [4], [5] to semantic retrieval [3], [6]. Lu et al., [4] expanded a query with synonyms obtained from WordNet and then performed keyword matching of method signatures. Lv et al., [5] expanded the query with the APIs and considered the impact of both text similarity and potential APIs on code search. Reiss et al., [3] developed a code retrieval system

[*]Equal contribution.

named Sourcerer, which learned the semantic representation of source code through a probabilistic topic model. Inspired by the success of deep learning in computer vision and natural language processing tasks, deep learning has been applied to better represent source code for tasks such as clone detection [7] and code summarization [8].

To the best of our knowledge, Gu et. al., [6] is the first who applied deep learning network to the task of code retrieval, which captures the correlation between the semantic source code and natural language query in an intermediate semantic space. However, the approach still suffers from two major limitations: (a) *Deep structured features of source code are often ignored*. The approach [6] captures the shallow source code information, including method name, code tokens and API sequence, missing the opportunity to capture the rich structure semantics of the code. (b) *Lack of explainability*. The final results from a deep neural network is often hard to interpret since its internal working is always transparent to input data and different applications. This is also a common issue when applying deep learning models. For example, in [6], the code and its natural language descriptions are projected into an intermediate semantic space and constrained by a ranking loss function. Although the semantic representation of code is learned, it is hard to infer which parts contribute more to the final result.

Insights. These aforementioned limitations motivate us to design a model which learns a more comprehensive representation on source code as well as with the ability of explainability. From one hand, for limitation (a), apart from the *tokens* of code, we also extract more features of code from its multiple views, such as *abstract syntax tree* (AST) and *control-flow graph* (CFG) [1]. The AST and CFG are two types of intermediate code, one of which represents the hierarchical syntactic structure of a program, and the other represents the computation and control flow of a program [9]. In this paper, we argue that aggregating complementary information from multiple views of source code can enrich its representation. In this paper, we use the term view and modality interchangeably. We call the approach of learning code representation from its multiple views/modalities as *multi-modal learning*. To address the limitation (b), since different modalities reflect different features of the source code. Therefore, each modality may not contribute equally to the final code representation. For a given modality, it consists of many elements (tokens, nodes in AST/CFG), weights are assigned to different elements via representation learning. Therefore, we can infer which part contributes more to the final result from the final representation, making explainability possible. In this paper, we design an attention mechanism to integrate the multi-modal features into a single hybrid code representation.

A Motivating Example. We give an example in Figure 1 to better illustrate our ideas. Figure 1(a) shows a simple C code example, which aims to verify whether an array of integers contains an even number. Figures 1(b) and (c) represent the corresponding AST and inter-procedural CFG of code in Figure 1(a), respectively. From Figure 1(a), we can see that the semantics of the highlighted three words `Verify`, `array`, `even` can be precisely captured by different code representations, e.g., plain text (for `check`), type-augmented AST (for `BinaryOperator`) and CFG (for `while`). These representations pay attention to different structure information of the code at different views, e.g., each node on AST represents a token and each node on CFG represents a statement. This shows the necessity of considering various modalities to better represent the source code. It is necessary to represent a code from multiple views, especially from the structured information, since the orders of tokens and statements on the two views can be different depending on different code representations. For example, based on plain text, the token after "`while`" in Figure 1 (a) is "`()`" and then followed by "`head`". Differently, on AST, there will be two possible tokens following "`Compound`", i.e., branch test "`if`", "`BinaryOperator`", as shown in Figure 1 (b). Similarly, after the token "}" in the last statement at line 6, there will be no token left based on plain text. However, based on CFG, the next token is "`while`" at the beginning of loop function based on CFG. From Figure 1, we can also observe that there exists an alignment relationship among the code snippet and it is description. For example, the keyword `Verify` should be closely connected to the word `check` in code. That means, on code retrieval, we can infer which part of the retrieved code contributes most to the input query words. This is very important to the model explainability.

Our Solution and Contributions. To tackle the two aforementioned issues, in this paper, we propose a novel model called Multi-Modal Attention Network (MMAN) for semantic source code retrieval. We not only consider the sequential features which have been studied in previous works (i.e., *method name* and *tokens*), but also the structure features (i.e., AST and CFG extracted from code). We explore a novel multi-modal neural network to effectively capture these multi-modal features simultaneously. In particular, we employ a LSTM [10] to represent the sequential tokens of code snippet, a Tree-LSTM [11] network to represent the abstract syntax tree (AST) and a gated graph neural network (GGNN) [12] to represent the CFG. To overcome the explainability issue, we design an attention mechanism to assign different weights to different parts of each modality of source code, with the ability of explanation. To summarize, the main contributions of this paper are as follows.

- We propose a more comprehensive multi-modal representation method for source code, with one LSTM for the sequential content of source code, a Tree-LSTM for the AST of source code and a GGNN for the CFG of source code. Furthermore, a multi-modal fusion layer is applied to integrate these three representations.
- To the best of our knowledge, it is the first time that we propose an attention network to assign different weights to different parts of each modality of source code, providing an explainability of our deep multi-modal neural network for representation.

[1]The tree structure can also be seen as a special instance of graph with no circles and with each node having at most one parent node.

- To verify the effectiveness of our proposed model, we validate our proposed model on a real-world dataset crawled from GitHub, which consists of $28,527$ C code snippets. Comprehensive experiments and analysis show the effectiveness of our proposed model when compared with some state-of-the-art methods.

Organization. The remainder of this paper is organized as follows. Section II highlights some works related to this paper. In Section III, we provide some background knowledge on multi-modal learning and attention mechanism. In Section IV, we first give an overview our proposed framework and then present each module of our proposed framework in detail. Section V describes the dataset used in our experiment and shows the experimental results and analysis. Section VI presents a discussion on our proposed model, including the strength as well as some threats to validity and limitations existing in our model. Finally, we conclude this paper and give some future research directions in Section VII.

II. Related Work

In this section, we briefly review the related studies from three perspectives, namely deep code representation, multi-modal learning and attention mechanism.

A. Deep Code Representation

With the successful development of deep learning, it has also become more and more prevalent for representing source code in the domain of software engineering research. In [13], Mou et al. learn distributed vector representations using tree-structured convolutional neural network (Tree-CNN) to represent snippets of code for program classification. Similarly, Wan et al. [8] apply the tree-structured recurrent neural network (Tree-LSTM) to represent the AST of source code for the task of code summarization. Piech et al. [14] and Parisotto et al. [15] learn distributed representations of source code input/output pairs and use them to guide program synthesis from examples. In [12], Li et al. represent heap state as a graph and proposed a gated graph neural network to directly learn its representation to mathematically describe the shape of the heap. Maddison and Tarlow [16] and other neural language models (e.g. LSTMs in Dam et al. [17]) describe context distributed representations while sequentially generating code. Ling et al. [18] and Allamanis et al. [19] combine the code-context distributed representation with distributed representations of other modalities (e.g., natural language) to synthesize code.

One limitation of the above mentioned approaches is that these approaches ignore CFG of source code, which also conveys rich semantic information. Furthermore, no unified network is proposed to effectively fuse these multiple modalities. To mitigate this issue, this papers resort to propose a multi-modal network to learn a more comprehensive representation of source code.

B. Multi-Modal Learning

One prevalent direction in multi-modal learning is on joint representation which has been applied in many applications such as image captioning [20], summarization [21], visual questioning answering [22] and dialog system [23]. In [20], Chen et al. propose an attentional hierarchical neural network to summarize a text document and its accompanying images simultaneously. In [21], Zhang et al. propose a multi-modal (i.e., image and long description of product) generative adversarial network for product title refinement in mobile E-commerce. In [22], Kim et al. propose a dual attention network to capture a high-level abstraction of the full video content by learning the latent variables of the video input, i.e., frames and captions. Similarly, in [23], Hori et al. answer questions about images using learned audio features, image features and video description, for the audio visual scene-aware dialog. Another direction in multi-modal learning is cross-modal representation learning for information retrieval, which is similar to our task. Cross-modal representation learning aims to learn representation of each modality via project them into an intermediate semantic space with a constraint. In [24], Carvalho et al. propose a cross-modal retrieval model aligning visual and textual data (like pictures of dishes and their recipes) in a shared representation space for receipt retrieval. In [25] Ma et al. propose a neural architecture for cross-modal retrieval, which combines one CNN for image representation and one CNN for calculating word-level, phrase-level and sentence-level matching scores between an image and a sentence. [26], [27], the authors learn hash functions that map images and text in the original space into a Hamming space of binary codes, such that the similarity between the objects in the original space is preserved in the Hamming space.

In this paper, we draw the insights from multi-modal learning, but not limit to it. We not only design a multi-modal neural network to represent the code, but also apply an attention mechanism to learn which part of code contributes more to the final semantic representation.

C. Attention Mechanism

Attention mechanism has shown remarkable success in many artificial intelligence domains such as neural machine translation [28], image captioning [29], image classification [30] and visual question answering [31]. Attention mechanisms allow models to focus on necessary parts of visual or textual inputs at each step of a task. Visual attention models selectively pay attention to small regions in an image to extract core features as well as reduce the amount of information to process. A number of methods have recently adopted visual attention to benefit image classification [32], [33], image generation [34], image captioning [35], visual question answering [36]–[38], etc. On the other hand, textual attention mechanisms generally aim to find semantic or syntactic input-output alignments under an encoder-decoder framework, which is especially effective in handling long-term dependency. This approach has been successfully applied to various tasks including machine translation [28], text generation [39], sentence summarization [8], [40], and question answering [41]. In [31], Lu et al. propose a co-attention learning framework to alternately learn the image attention and the question attention for visual question

answering. In [42], Nam et al. propose a multi-stage co-attention learning framework to refine the attentions based on memory of previous attentions. In [43], Paulus et al. combine the inter- and intra-attention mechanism in a deep reinforcement learng setting to improve the performance abstractive text summarization. In [44], Zhang et al. introduce a self-attention mechanism into convolutional generative adversarial networks.

To the best of our knowledge, no study has attempted to learn multimodal attention models for the task of code retrieval.

III. Preliminaries

In this section, we first mathematically formalize the code retrieval problem using some basic notations and terminologies. We then present some background knowledge of multi-modal learning and attention mechanism.

A. Problem Formulation

To start with, we introduce some basic notations. Suppose that we have a set $\mathcal{D}$ of N code snippets, with corresponding descriptions, i.e., $\mathcal{D} = \{< x_1, d_1 >, < x_2, d_1 >, \ldots, < x_N, d_N >\}$. Each code snippet and description can be seen as a sequence of tokens. Let $x_i = (x_{i1}, x_{i2}, \ldots, x_{i|x_i|})$ be a sequence of source code snippet, $d_i = (d_{i1}, d_{d2}, \ldots, d_{i|d_i|})$ be a sequence of a description, where $|\cdot|$ denotes the length of a sequence. As we declared before, we represent the source code from three modalities (i.e., tokens, AST and CFG). We denote the semantic representation of code snippet x_i as $\mathbf{x}_i =< \mathbf{x}_i^{tok}, \mathbf{x}_i^{ast}, \mathbf{x}_i^{cfg} >$, where $\mathbf{x}_i^{tok}$, $\mathbf{x}_i^{ast}$, $\mathbf{x}_i^{cfg}$ denote the representation for the three modalities, respectively.

Since the code snippet and its description are heterogeneous, the goal of this paper is to train a model to learn their representation in an intermediate semantic space, simultaneously. Then, in the testing phase, the model can return a similarity vector of each candidate code snippet for a given query.

B. Multi-Modal Learning

Multi-modal learning aims to build models that can process and aggregate information from multiple modalities [45]. One important task of multi-modal learning is multi-modal representation learning, which is roughly categorized as two classes: *joint* and *coordinated*. Joint representations combine the unimodal signals into the same representation space, while coordinated representations process unimodal signals separately, but enforcing certain similarity constraints on them to bring them to what we term an intermediate semantic space. We introduce these two kinds of techniques in our problem setting. Figure 2 illustrates the difference and connection between of *joint* and *coordinated* representations.

For code snippet $\mathbf{x}$, we extract its multiple modalities such as $\mathbf{x}^{tok}$, $\mathbf{x}^{ast}$, $\mathbf{x}^{cfg}$. Since these modalities are complementary representation of a same code, we can apply the joint representation, which is formularized as follows:

$$\mathbf{x} = f\left(\mathbf{x}^{tok}, \mathbf{x}^{ast}, \mathbf{x}^{cfg}\right), \tag{1}$$

where the multimodal representation $\mathbf{x}$ is computed using function f (e.g., a deep neural network) that relies on unimodal representations $\mathbf{x}^{tok}, \mathbf{x}^{ast}, \mathbf{x}^{cfg}$.

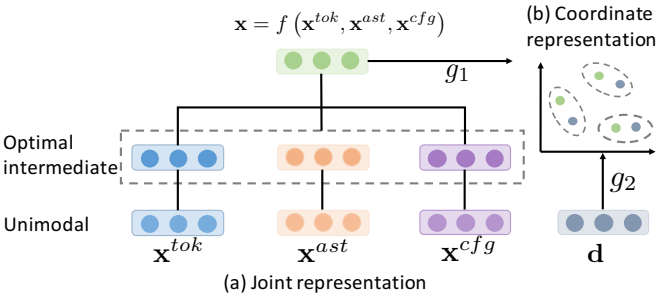


Figure 2: The difference and connection between *joint* and *coordinated* representations (adapted from [45]).

While considering the code snippet $\mathbf{x}$ and description $\mathbf{d}$, since these two modalities are from different sources, it is desirable for us to apply coordinated representation for them [6], which is defined as follows:

$$g_1\left(\mathbf{x}\right) \sim g_2\left(\mathbf{d}\right), \tag{2}$$

where each modality has a corresponding projection function (g_1 and g_2 above) that projects it into an intermediate semantic space with a similarity constraint/coordination on them. Examples of such coordination include minimizing cosine distance, or maximizing correlation. In this paper, the cosine similarity function is adopted.

C. Attention Mechanism

Attention networks learn functions that provide a weighting over inputs or internal features to steer the information visible to other parts of a network. To some extent, it is biologically motivated by the fact that our retina pays visual attention to different regions of an image or correlate words in one sentence. To date, many variants of attention mechanism have been evolved. From another perspective, the attention mechanism can be seen as the process of soft-addressing in a memory unit. The source, composed of key $\mathbf{k}$ and value $\mathbf{v}$, can be seen as the content of memory. Given an input query, the goal of attention mechanism is to return an attention value. Formally, we can define the attention value among query, key and value as follows.

$$\alpha(\mathbf{q}, \mathbf{k}) = \text{softmax}(g(\mathbf{q}, \mathbf{k})), \tag{3}$$

where $\mathbf{q}$ is the query and $\mathbf{k}$ is the key, g is the attention score function which measures the similarity between query and key. Usually, the g has many options, such as multi-layer perceptron [28], dot product [46] and scaled dot product [47]. We call this kind of attention as inter-attention. However, there exists a condition that the query is the key itself. In this condition, we call it intra-attention (also well known as self-attention) [48], [49], which exhibits a better balance between ability to model long-range dependencies and computational and statistical efficiency. After obtaining the attention score, the final attended vector can be represented as the weighted sum of each value in the memory:

$$\mathbf{v} = \sum_i \alpha_i(\mathbf{q}, \mathbf{k})\mathbf{v}_i, \tag{4}$$

where $\mathbf{v}_i$ is the i-th value in the memory. In this paper, we adopt the inter-attention. Furthermore, the key is the hidden state of token or node in AST/CFG, and the value is the corresponding context vector (cf. Sec. IV-C).

IV. Multi-Modal Attention Network

A. An Overview

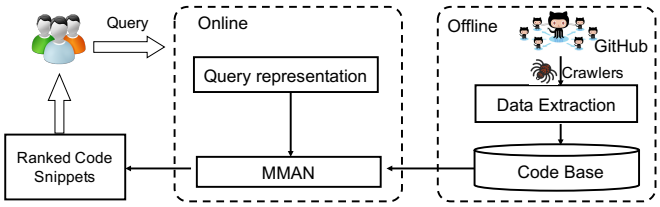


Figure 3: The workflow of MMAN.

In this section, we firstly give an overall workflow of how to get a trained model for code retrieval. Then we present an overview of the network architecture of our proposed MMAN model.

Figure 3 shows the overall workflow of how to get a trained model, which consists of an offline training stage and an online retrieval stage. In the training stage, we prepare a large-scale corpus of annotated <code, description> pairs. The annotated pairs are then fed into our proposed MMAN model for training. After training, we can get a trained retrieval network. Then, given a natural language query, related source code snippets can be retrieved by the trained network.

Figure 4 is an overview of the network architecture of our proposed MMAN model. We split the framework into three submodules. (a) Multi-modal code representation (cf. Sec. IV-B). This module is used to represent the source code into a hidden space. (b) Multi-modal attention fusion (cf. Sec. IV-C). This attention module is designed to assign different weight on different parts for each modality, and then fuse the attended vector into a single vector. (c) Model learning (cf. Sec. IV-E). This attention module is designed to learn the comment description representation and code representation in a common space through a ranking loss function. We will elaborate each component in this framework in the following sections.

B. Multi-Modal Code Representation

Different from previous methods that just utilize sequential tokens to represent code, we also consider the structure information of source code, in this section, we present a hybrid embedding approach for code representation. We apply a LSTM to represent the tokens of code, and a Tree-LSTM to represent the AST of code a GGNN to represent the CFG of code.

1) Lexical Level - Tokens: The key insight into lexical level representation of source code is that the comments are always extracted from the lexical of code, such as the method name, variable name and so on. In this paper, we apply LSTM network to represent the sequential tokens.

$$\mathbf{h}_i^{tok} = \text{LSTM}\left(\mathbf{h}_{i-1}^{tok}, w(x_i)\right), \tag{5}$$

where $i = 1, \ldots, |x|$, w is the word embedding layer to embed each word into a vector. The final hidden state $\mathbf{h}_{|x|}^{tok}$ of the last token of code is the token modality representation of x.

2) Syntactic Level - AST: We represent the syntactic level of source code from the aspect of AST embedding. Similar to a traditional LSTM unit, we propose Tree-LSTM where the LSTM unit also contains an input gate, a memory cell and an output gate. However, different from a standard LSTM unit which only has one forget gate for its previous unit, a Tree-LSTM unit contains multiple forget gates. In particular, considering a node N with the value x_i in its one-hot encoding representation, and it has two children N_L and N_R , which are its left child and right child, respectively. The Tree-LSTM recursively computes the embedding for N from the bottom up. Assume that the left child and the right child maintain the LSTM state $(\mathbf{h}_L, \mathbf{c}_L)$ and $(\mathbf{h}_R, \mathbf{c}_R)$, respectively. Then the LSTM state (h, c) of N is computed as

$$(\mathbf{h}_i^{ast}, \mathbf{c}_i^{ast}) = \text{LSTM}\left(\left(\left[\mathbf{h}_{iL}^{ast}; \mathbf{h}_{iR}^{ast}\right], \left[\mathbf{c}_{iL}^{ast}; \mathbf{c}_{iR}^{ast}\right]\right), w(x_i)\right), \tag{6}$$

where $i = 1, \ldots, |x|$ and $[\cdot;\cdot]$ denotes the concatenation of two vectors. Note that a node may lack one or both of its children. In this case, the encoder sets the LSTM state of the missing child to zero. In this paper, we adopt the hidden state of root node as the AST modality representation. It's worth mentioning that when the tree is simply a chain, namely $N = 1$, the Tree-LSTM reduces to the vallina LSTM. Figure 5 shows the structure of LSTM and Tree-LSTM.

3) Syntactic Level - CFG: As the CFG is a directed graph, we apply a gated graph neural network (GGNN) to represent the CFG, which is a neural network architecture developed for graph. We first define a graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is a set of vertices (v, ℓ_v) and $\mathcal{E}$ is a set of edges (v_i, v_j, ℓ_e). ℓ_v and ℓ_e are labels of vertex and edge, respectively. In our code retrieval scenario, each vertex is the node of CFG, and each edge represents the control-flow of code, which has multiple types. GGNN learns the graph representation directly through the following procedures: First, we initialize the hidden state for each vertex $v \in \mathcal{V}$ as $\mathbf{h}_{v,0}^{cfg} = w(\ell_v)$, where w is the one-hot embedding function. Then, for each round t, each vertex $v \in \mathcal{V}$ receives the vector $\mathbf{m}_{v,t+1}$, which is the "message" aggregated from its neighbours. The vector $\mathbf{m}_{v,t+1}$ can be formulated as follows:

$$\boldsymbol{m}_{v,t+1} = \sum_{v' \in \mathcal{N}(v)} \mathbf{W}_{\ell_e} \mathbf{h}_{v',t}, \tag{7}$$

where $\mathcal{N}(v)$ are the neighbours of vertex v. For round t, message from each neighbour is mapped into a shared space via $\mathbf{W}_{\ell_e}$.

For each vertex $v \in \mathcal{V}$, the GGNN update its hidden state with a forget gate. In this paper, we adopt the gated recurrent unit (GRU) [50] to update the hidden state of each vertex, which can be formulated as follows.

$$\mathbf{h}_{v,t+1}^{cfg} = \text{GRU}(\mathbf{h}_{v,t}^{cfg}, \mathbf{m}_{v,t+1}). \tag{8}$$

Finally, with T rounds of iterations, we aggregate the hidden states of all vertices via summation to obtain the embedded

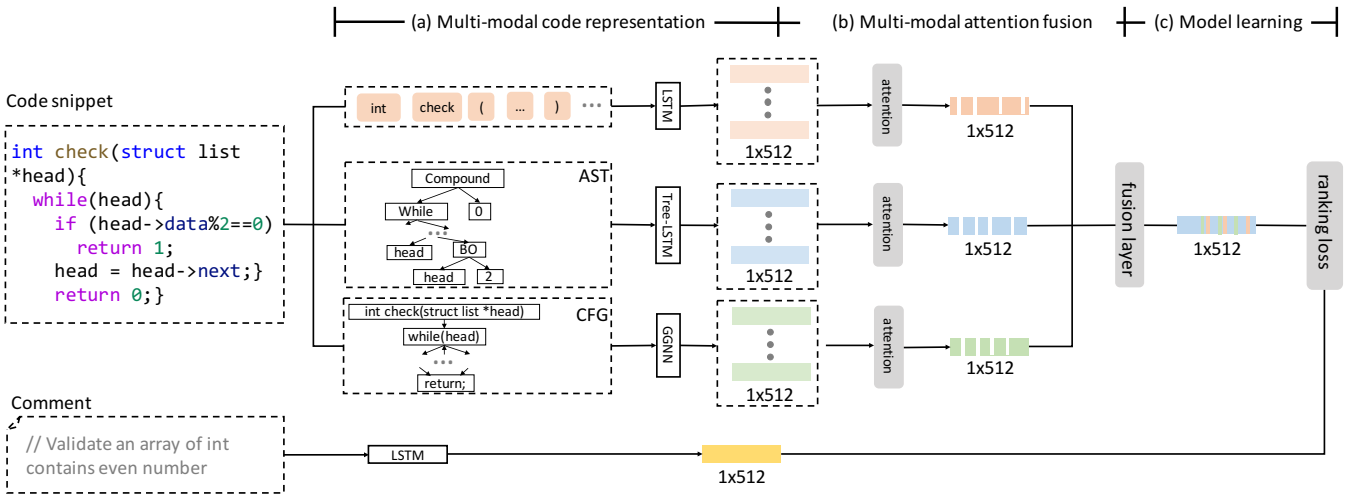


Figure 4: The network architecture of our proposed MMAN model. We first extract the <code, description> pairs from training corpus. We then parse the code snippets into tree modalities, i.e., tokens, AST, CFG. Then the training samples are fed into the network as input. (a) Multi-modal code representation. We first learn the representation of each modality via LSTM, Tree-LSTM and GGNN, respectively. (b) Multi-modal attention fusion. We design an attention layer to assign different weight on different parts for each modality, and then fuse the attended vector into a single vector. (c) Model learning. We map the comment description representation and code representation into an intermediate semantic common space and design a ranking loss function to learn their similarities.

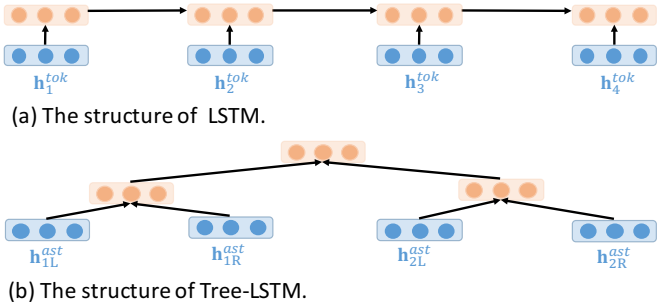


Figure 5: An illustratation of Tree-LSTM and LSTM.

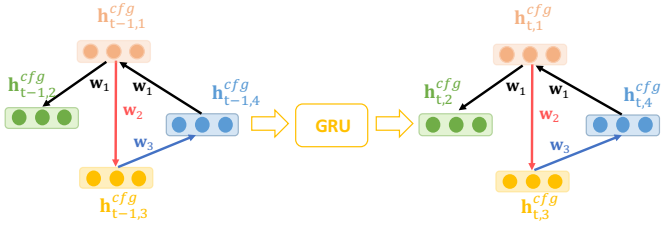


Figure 6: An illustration of GGNN.

representation of the CFG. Figure 6 illustrates the structure of GGNN.

C. Multi-Modal Attention Fusion

After we obtain the semantic representation of each modality, we need to fuse them into a single representation. As we declare before, for a unimodal, since it is composed of many elements, it is desirable to assign different weights to each element.

Token Attention. For tokens, not all tokens contribute equally to the final semantic representation of code snippet. Therefore, we introduce the attention mechanism on tokens to extract the ones that are more important to the representation of a sequence of code tokens. The attention score for tokens α^{tok} is calculated as follows:

$$\alpha_i^{tok} = \frac{\exp(g^{tok}(f^{tok}(\mathbf{h}_i^{tok}), \mathbf{u}^{tok}))}{\sum_j \exp(g^{tok}(f^{tok}(\mathbf{h}_j^{tok}), \mathbf{u}^{tok}))}, \tag{9}$$

where $\mathbf{h}_i^{tok}$ represents the hidden state of i-th token in code. f^{tok} denotes a linear layer and g^{tok} is the dot-product operator. $\mathbf{u}^{tok}$ is the context vector of token modality, which can be seen as a high level representation of sequential tokens of code. The word context vector $\mathbf{u}^{tok}$ is randomly initialized and jointly learned during the training process.

AST Attention. For the AST, not all nodes contribute equally to the final semantic representation of code snippet, indicating that different construct occurring in the source code (e.g., `if-condition-then`) should also be considered distinctly. Similar to Token attention, the attention score for AST nodes α^{ast} is calculated as follows:

$$\alpha_i^{ast} = \frac{\exp(g^{ast}(f^{ast}(\mathbf{h}_i^{ast}), \mathbf{u}^{ast}))}{\sum_j \exp(g^{ast}(f^{ast}(\mathbf{h}_j^{ast}), \mathbf{u}^{ast}))}, \tag{10}$$

where $\mathbf{h}_i^{ast}$ represents the hidden state of i-th node in the AST. f^{ast} denotes a linear layer and g^{ast} is the dot-product operator. $\mathbf{u}^{ast}$ is the context vector of AST modality, which can be seen as a high level representation of AST nodes of code.

CFG Attention. For the CFG, different statement in the source code should also be assigned different weight for the final

representation. Therefore, we assign each CFG nodes with the weight α^{cfg} as:

$$\alpha_i^{cfg} = \text{sigmoid}(g^{cfg}(f^{cfg}(\mathbf{h}_i^{cfg}), \mathbf{u}^{cfg}), \tag{11}$$

where $\mathbf{h}_i^{cfg}$ represents the hidden state of i-th node in the CFG. f^{cfg} denotes a linear layer and g^{cfg} is the dot-product operator. $\mathbf{u}^{cfg}$ is the context vector of CFG modality, which can be seen as a high level representation of CFG nodes of code. It's worth mentioning that CFG attention weighted by sigmoid function achieves better performance than that by softmax function from the experimental results.

Multi-Modal Fusion. We then integrate the multi-modal representation into a single representation via their corresponding attention score. We first concatenate them and then feed them into a one-layer liner network, which can be formularized as follows.

$$\mathbf{x} = \mathbf{W}\left[\sum_i \alpha_i^{tok}\mathbf{h}_i^{tok}; \sum_i \alpha_i^{ast}\mathbf{h}_i^{ast}; \sum_i \alpha_i^{cfg}\mathbf{h}_i^{cfg}\right], \tag{12}$$

where $\mathbf{x}$ is the final semantic representation of code snippet x, $[\cdot;\cdot]$ is the concatenation operation and $\mathbf{W}$ is the attention weight for each modality.

D. Description Representation

In the training phase, the descriptions are extracted from the code comments, while in the testing phase, the description are regarded as the input queries. In this paper, we apply a vallina LSTM to represent the description.

$$\mathbf{h}_i^{des} = \text{LSTM}\left(\mathbf{h}_{i-1}^{des}, w(d_i)\right), \tag{13}$$

where $i = 1, \ldots, |d|$ and w is the word embedding layer to embed each word into a vector. The hidden state of last step $\mathbf{h}_{|d|}^{des}$ can be used as a vector representation of d.

E. Model Learning

Now we present how to train the MMAN model to embed both code and descriptions into an intermediate semantic space with a similarity coordination. The basic assumption of this joint representation is that if a code snippet and a description have similar semantics, their embedded vectors should be close to each other. In other words, given an arbitrary code snippet x and an arbitrary description d, we want it to predict a high similarity if d is a correct description of x, and a small similarity otherwise. In training phase, we construct each training instance as a triple $< x, d^+, d^- >$: for each code snippet x, there is a positive description d^+ (a correct description of x) as well as a negative description (an incorrect description of x) d^- randomly chosen from the pool of all d^+'s. When trained on the set of $< x, d^+, d^- >$ triples, the MMAN predicts the cosine similarities of both $< x, d^+ >$ and $< x, d^- >$ pairs and minimizes the hinge range loss [6], [51]:

$$\mathcal{L}(\theta) = \sum_{<x,d^+,d^->\in\mathcal{D}} \max(0, \beta - sim(\mathbf{x}, \mathbf{d}^+) + sim(\mathbf{x}, \mathbf{d}^-)), \tag{14}$$

where θ denotes the model parameters, $\mathcal{D}$ denotes the training dataset, sim denotes the similarity score between code and description β is a small constant margin. $\mathbf{x}$, $\mathbf{d}^+$ and $\mathbf{d}^-$ are the embedded vectors of x, d^+ and d^-, respectively. In our experiments, we adopt the cosin similarity function (cf. IV-F) and set the fixed β value to 0.05. Intuitively, the ranking loss encourages the similarity between a code snippet and its correct description to go up, and the similarities between a code snippet and incorrect descriptions to go down.

F. Code Retrieval

After the model is trained, we can deploy it online for service. Given a code base $\mathcal{X}$, for a given query q, the target is to rank all these code snippets by their similarities with query q. We first feed the code snippet x into the multi-model representation module and feed the query q as description into the LSTM module to obtain their corresponding representations, denoted as $\mathbf{x}$ and $\mathbf{q}$. Then we calculate the ranking score as follows:

$$sim(x, q) = \cos(\mathbf{x}, \mathbf{q}) = \frac{\mathbf{x}^T\mathbf{q}}{\|\mathbf{x}\|\|\mathbf{q}\|}, \tag{15}$$

where $\mathbf{x}$ and $\mathbf{q}$ are the vectors of code and a query, respectively. The higher the similarity, the more related the code is to the given query.

V. Experiments and Analysis

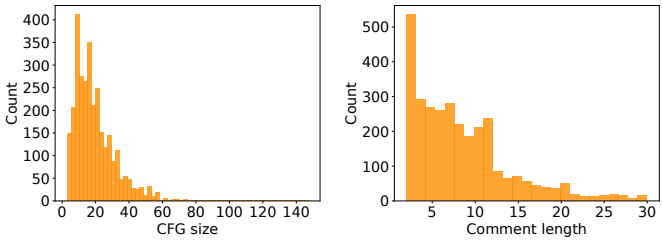
To evaluate our proposed approach, in this section, we conduct experiments to answer the following questions:

- ***RQ1.*** Does our proposed approach improve the performance of code retrieval when compared with some state-of-the-art approaches?
- ***RQ2.*** What is the effectiveness and the contribution of each modality of source code, e.g., sequential tokens, AST, CFG of source code for the final retrieval performance, and what about their combinations?
- ***RQ3.*** What is the performance of our proposed model when varying the code length, code length, code AST node number, code CFG node number and comment length?
- ***RQ4.*** What is the performance of our proposed attention mechanism? What is the explainability of attention visualization?

We ask RQ1 to evaluate our deep learning-based model compared to some state-of-the-art baselines, which will be described in the following subsection. We ask RQ2 in order to evaluate the performance of each modality extracted from source code. We ask RQ3 to analyze the sensitivity of our proposed model when varying the code length, code AST node number, code CFG node number and comment length. We ask RQ4 to verify the explainability of our proposed attention mechanism. In the following subsections, we first describe the dataset, some evaluation metrics and the training details. Then, we introduce the baseline for RQ1. Finally, we report our results and analysis for four research questions.

A. Dataset Collection

As described in Section IV, our proposed model requires a large-scale training corpus that contains code snippets and their corresponding descriptions. Following but different from [6],



(a) CFG node number. (b) Comment length.

Figure 7: The histogram of the dataset in our experiments. (a) CFG node number distribution. (b) Comment length distribution.

we evaluate the performance of our proposed model on a corpus of C code snippets, collected from GitHub (a popular open source projects hosting platform). Actually, we have considered the dataset released by [6], while this dataset only contains the cleaned Java snippets without the raw data, unable to generate the CFG. Therefore, we resort to build a more complicated language C dataset, which may also provide more challenges and opportunities for our further research.

To construct the codebase, we crawl all the C language repositories by its API[2]. We then exclude the repositories whose stars number is smaller than 2. We select only the methods that have documentation comments from the crawled projects. Finally, we obtain a C corpus consisting of $28,527$ commented C methods.

Figure 7 shows the length distributions of code and comment on testing data. From Figure 7a, we can find that the lengths of most code snippets are located between 20 to 40. This was also observed in the quote in [52] "Functions should hardly ever be 20 lines long". From Figure 7b, we can notice that the lengths of nearly all the comments are smaller than 10. This also confirms the challenge for capturing the correlation between short text with its corresponding code snippet.

Having collected the corpus of commented code snippets, we extract the multi-modal code features and it's corresponding description, i.e., <method name, tokens, AST, CFG, description>, as follows:

Method Name Extraction. For each C method, we extract its name and parse the name into a sequence of tokens according to camel case, and if it contains _, we then tokenize it via _.

Token Extraction. To collect tokens from a C method, we tokenize the code by {`. , " ; ) ( ! (space)`}. After we tokenize function body, function name, we limit their max length as 100 and 50 respectively.

AST Extraction. To construct the tree-structure of code, we parse C code into abstract syntax trees via an open source tool named Clang (`http://clang.llvm.org/`). For simplification, we transform the generated ASTs to binary trees by the following two steps which have been adopted in [8]: a) split nodes with more than 2 children, generate a new right child together with the old left child as its children, and then put all children except the leftmost as the children of this new node. Repeat this operation in a top-down way until only nodes with 0, 1, 2 children left; b) combine nodes with 1 child with its child.

CFG Extraction. To construct the CFG of code, we first parse C function into CFG via an open source tool named SVF [53] (`https://github.com/SVF-tools/SVF`), which has been widely used in value-flow analysis [54], [55]. We then remove nodes with same statement or no statement. For nodes with same statement, we retain the nodes which occur in the output of SVF first and remove their child nodes, and link children of their child nodes to them. For nodes without statement, we delete them and link their child nodes to their parent nodes. We set maximum size of CFG nodes as 512.

Description Extraction. To extract the documentation comment, we extract description via the regular expression `/**/`. We check the last sentence before every function and if it meets the condition that we have defined via regular expression, then we extract the description from the sentence.

We shuffle the dataset and split it into two parts, namely 27,527 samples for training and 1,000 samples for evaluation. It's worth mentioning a difference between our data processing and the one in [6]. In [6], the proposed approach is verified on another isolated dataset to avoid the bias. Since the evaluation dataset doesn't have the ground truth, they manually labeled the searched results. We argue that this approach may introduce the human bias. Therefore, in our paper, we resort to the automatic evaluation.

B. Evaluation Metrics

For automatic evaluation, we adopt two common metrics to measure the effectiveness of code retrieval, i.e., $\mathrm{SuccessRate}@k$ and Mean Reciprocal Rank (MRR), both of which have been widely used in the area of information retrieval. To measure the relevance of our search results, we use the success rate at rank k. The $\mathrm{SuccessRate}@k$ measures the percentage of queries for which more than one correct result could exist in the top k ranked results [6], [56], which is calculated as follows:

$$\mathrm{SuccessRate}@k = \left(\frac{1}{|Q|} \sum_{q=1}^{Q} \delta\left(\mathrm{FRank}_q \leq k\right) \right), \qquad (16)$$

where Q is a set of queries, $\delta(\cdot)$ is a function which returns 1 if the input is true and returns 0 otherwise. $\mathrm{SuccessRate}@k$ is important because a better code search engine should allow developers to discover the needed code by inspecting fewer returned results. The higher the $\mathrm{SuccessRate}$ value is, the better the code search performance is.

We also use Mean Reciprocal Rank (MRR) to evaluate the ranking of our search results. The MRR [5], [6] is the average of the reciprocal ranks of results of a set of queries Q. The

[2]We crawled the GitHub in Oct., 2016, so the repositories in our database are created from August, 2008 to Oct., 2016.

reciprocal rank of a query is the inverse of the rank of the first hit result [6]. The MRR is defined as

$$\text{MRR} = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{\text{FRank}_q}, \tag{17}$$

where $|Q|$ is the size of query set. The higher the MRR value is, the better the code retrieval performance is.

C. Implementation Details

To train our proposed model, we first randomize the training data and set the mini-batch size to 32. We build three separate vocabulary for code, comment and AST leaf node tokens with size 19,665, 9,628 and 50,004, respectively. For each batch, the code is padded with a special token `<PAD>` to the maximum length. All tokens in our dataset are converted to lower case. We set the word embedding size to 300. For LSTM and Tree-LSTM unit, we set the hidden size to be 512. For GGNN, we set the hidden size to 512 and set 5 rounds of iteration for GGNN. The margin β is set to 0.05. We update the parameters via Adam [57] optimizer with the learning rate 0.0001. To prevent over-fitting, we use dropout with 0.1. All the models in this paper are trained for 100 epochs, taking about 41 hours. All the experiments are implemented using the PyTorch 1.2 framework with Python 3.7, and the experiments were conducted on a computer with a Nvidia Quadro RTX 8000 GPU with 48 GB memory, running Ubuntu 18.04.

D. Q1: Comparison with Baselines

We compare our model with the following baseline methods:

- CodeHow [5] is a state-of-the-art code search engine proposed recently. It is an information retrieval based code search tool that incorporates an extended Boolean model and API matching.
- DeepCS [6] is the state-of-the-art deep neural network based approach for code retrieval. It learns a unified vector representation of both source code and natural language queries.
- MMAN (Tok/AST/CFG)-w/o.Att. represents our proposed model on three modalities, without attention component for each modality. We also derive its variants on composition of these modalities.
- MMAN (Tok/AST/CFG)-w.Att. represents our proposed model on three modalities, with attention component for each modality. We also derive its variants on combinations of these modalities.

Automatic Evaluation. We evaluate MMAN on the evaluation dataset, consisting of $1,000$ descriptions. In this automatic evaluation, we consider each description as an input query, and its corresponding code snippet as the ground truth. Table I shows the overall performance of the three approaches, measured in terms of $SuccessRate@k$ and MRR. The columns $R@1$, $R@5$ and $R@10$ show the results of the average $SuccessRate@k$ over all queries when k is 1, 5 and 10, respectively. The column MRR shows the MRR values of the three approaches. From this table, we can draw the following conclusions: (a) Under all experimental settings, our MMAN (Tok+AST+CFG) method obtains higher performance in terms of both metrics consistently, which indicates better code retrieval performance. For the $R@k$, the improvements to DeepCS are 26.18%, 16.06% and 12.21%, respectively. For the MRR, the improvement to DeepCS is 19.89%. (b) Comparing the performance of two variants of our proposed model MMAN (Tok+AST+CFG)-w. or w/o.Att., we can observe that our designed attention mechanism indeed has a positive effect.

Table I: Comparison of the overall performance between our model and baselines on automatic evaluation metrics. (Best scores are in boldface.)

Method	**R@1**	**R@5**	**R@10**	**MRR**
CodeHow	0.262	0.474	0.543	0.360
DeepCS	0.275	0.498	0.565	0.377
MMAN (Tok+AST+CFG)-w/o.Att.	0.243	0.460	0.517	0.343
MMAN (Tok+AST+CFG)-w.Att.	**0.347**	**0.578**	**0.634**	**0.452**

Table II: Effect of each modality. (Best scores are in boldface.)

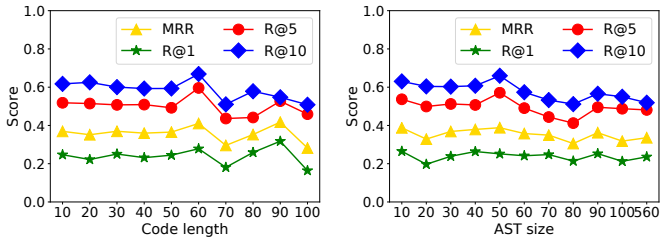
Method	**R@1**	**R@5**	**R@10**	**MRR**
MMAN (Tok)-w/o.Att.	0.277	0.468	0.521	0.365
MMAN (AST)-w/o.Att.	0.273	0.458	0.511	0.358
MMAN (CFG)-w/o.Att.	0.110	0.286	0.363	0.193
MMAN (Tok+AST+CFG)-w/o.Att.	0.243	0.46	0.517	0.343
MMAN (Tok)-w.Att.	0.327	0.562	0.619	0.432
MMAN (AST)-w.Att.	0.288	0.496	0.542	0.379
MMAN (CFG)-w.Att.	0.119	0.303	0.387	0.204
MMAN (Tok+AST+CFG)-w.Att.	**0.347**	**0.578**	**0.634**	**0.452**

E. Q2: Effect of Each Modality

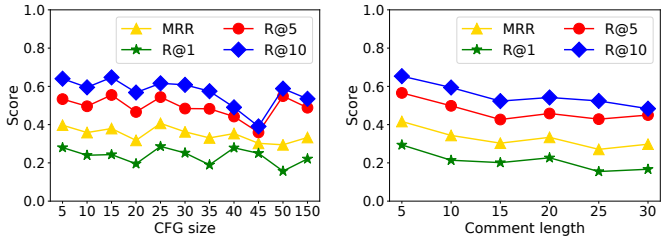
To demonstrate the effectiveness of fusing multiple modalities, we have conducted experiments over different modality combinations to validate the effectiveness of fusing multiple modalities. Table II presents the performance of MMAN over various source combinations with and without attention. From this table, we can observe that incorporating more modalities will achieve a better performance, which shows that there is a complementary rather than conflicting relationship among these modalities. In a sense, this is consensus with the old saying "two heads are better than one". Comparing the performance of each modality comparison with or without attention, we also obtain that our designed attention mechanism has a positive effect on fusing these modalities together.

F. Q3: Sensitivity Analysis

To analyze the robustness of our proposed model, we study four parameters (i.e., code length, code AST node number, code CFG node number and comment length) which may have an effect on the code and comment representation. Figure 8a shows the performance of our proposed method based on different evaluation metrics with varying code and comment lengths. From Figures 8a (a)(b)(c), we can see that in most cases, our proposed model has a stable performance even though the code length or node number increases dramatically. We take this effect into account by attributing it to the hybrid representation we adopt in our model. From Figure 8a(d), we can see that the performance of our proposed model decreases



(a) Code length. (b) AST node number.



(c) CFG node number. (d) Comment length.

Figure 8: Experimental results of our proposed method on different metrics w.r.t. varying code CFG node number and comment length.

on four metrics when the lengths of comments increase. This shows that increasing the length of a comment will increase the difficulty of comment understanding. Overall, it further verifies the robustness of multi-modal code representation.

G. Q4: Qualitative Analysis and Visualization

In the previous sections, we have shown the effectiveness of our proposed attention mechanism from the evaluation metrics. To gain some insights on the superiority of multi-modal representation and how the attention works, we show some cases retrieved by our model with different modality combinations (see Figure 9) and a visualization on the attention of each modality to interpret which part of code contributes more to the final result (see Figure 10).

Figure 9 shows the first retrieved result of our proposed MMAN model with different modality combination and the DeepCS for query "*Print any message in the axel structure*". We can see that our proposed model can accurately retrieve the ground truth code snippets, when compared with the state-of-the-art DeepCS model (as shown in Figure 9b). When comparing Figure 9a with Figure 9c and Figure 9d, we can also clearly see the superiority of multi-modal approach for accurate code representation.

Figure 10 visualizes the attention weights assigned to each part of code w.r.t. different modalities, for the sake of interpreting which part of code contributs more to the final result. From Figure 10a, we can see that the attention on tokens can really extract the important parts such as the function name `print_message`. On the other hand, from Figure 10b, we can see the attention on AST assigns more weight on the leaf node (e.g., `axel`) as well as some operation nodes (e.g., `BinaryOperator`). Furthermore, from Figure 10c, we observe that the attention on CFG assigns more weight on the invoked function node (e.g., `prinf` and `free`). This can be illustrated by the fact that CFG can describe the control flow of a source code snippet.

```
void print_messages(axel_t*axel)
{
  message_t *m;
  while(axel->message){
  printf("%s\n",axel->message->text);
  m=axel->message;
  axel->message=axel->message->next;
  free(m);
}}
```

(a) MMAN (Tok+AST+CFG) w. Attn

```
void printOutputCode(FILE *out, unsigned int
word, int word_len){
unsigned int tmp;
int i;
for(i=word_len-1; i>=0; i--){
  tmp = word;
  tmp = 1 & (tmp>>i);
  if(tmp) fputc('1', out);
  else fputc('0', out);}
fputc('\n', out);}
```

(b) DeepCS

```
void printList(struct node *node)
{
  while(node!=NULL)
  {
    printf("%s ", node->data);
    node = node->next;
  }
}
```

(c) MMAN (AST) w. Attn

```
static void
json_parse_array_context(lua_State *l,
json_parse_t *json) {
json_token_t token; int i;
json_decode_descend(l, json, 2);
lua_newtable(l);
json_next_token(json, &token);
if (token.type == T_ARR_END){
  json_decode_ascend(json);return;}
for (i = 1; ; i++) {
  ...
  if (token.type != T_COMMA)
    ...
}}
```

(d) MMAN (CFG) w. Attn

Figure 9: The retrieved first result of our proposed MMAN model with different modality combination and the DeepCS for query "*Print any message in the axel structure*"

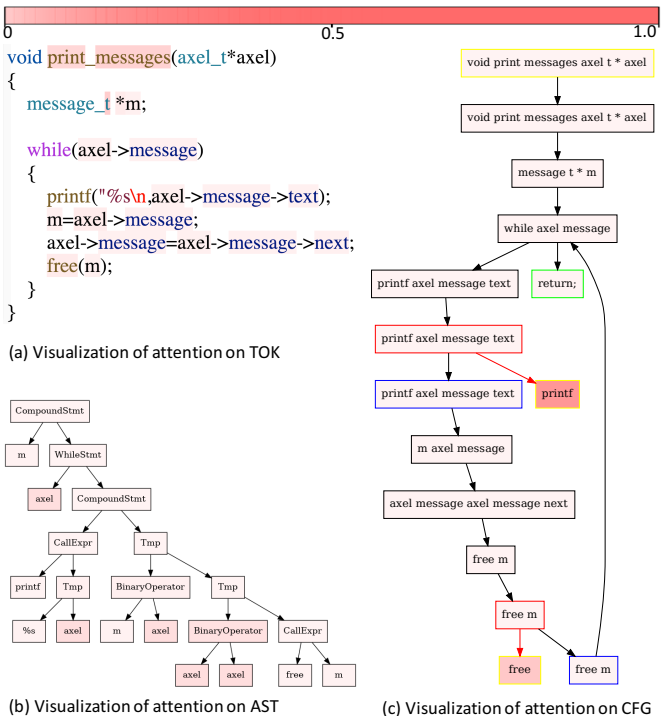


Figure 10: Attention visualization to interpret the contribution of each part of code w.r.t. different modalities.

VI. Discussion

A. Strength of MMAN

We have identified three advantages of MMAN that may explain its effectiveness in code retrieval: (a) *A more compresensive representation of source code from its multiple modalities.* The MMAN represents the source code snippet from its multiple modalities (i.e., tokens, AST and CFG) which contains complementary information for code representation. (b) *An attention mechanism to assign different weights on different parts for each modality.* The MMAN contains an attention mechanism which can infer the contribution of each part of code to the final result, and through visualization, we can obtain a explainability for our deep learning based model, and (c) *A unified framework to learn the heterogeneous representation of source code and description in an intermediate semantic space.* The MMAN is an end-to-end neural network model with a unified architecture to learn the representation of source code and description in an intermediate semantic space.

B. Threats to Validity and Limitations

Our proposed MMAN may suffer from two threats to validity and limitations. One threat to validity is on the evaluation metrics. We evaluate our approach using only two metrics, i.e., $SuccessRate@R$ and MRR, which are both standard evaluation metrics in information retrieval. We do not use precision at some cutoff ($Precision@k$), since the relevant results need to be labelled manually. We argue that this kind of approach will introduce the human bias. However, a human evaluation is also needed for the sake of fair comparison with DeepCS.

Another threat to validity lies in the extensibility of our proposed approach. Our model needs to be trained on a large scale of corpus, which is collected from online platforms such as GitHub. Since the writing style of different programmers may differ greatly, lots of efforts will be put into this step. In this paper, we have defined many regular expressions to extract the samples that meets our condition, at the same time, many samples are filtered. Furthermore, the CFG can only be extracted from a whole program. Therefore, it's difficult to extend our multi-modal code representation model to some contexts where the CFG are unable to be extracted, such as many code snippets from StackOverflow.

VII. Conclusion and Future Work

In this paper, we have proposed a novel multi-modal neural network with attention machanism named MMAN for the task of code retrieval. Apart from considering the sequential features of source code such as *tokens* and *API sequences*, MMAN also considers the structure features of code such as AST and CFG to learn a more comprehensive semantic representation for code understanding. Furthermore, we put forward an attention mechanism to interpret the contribution of each part of code. In addition, we proposed a unified framework to learn the representation of code representation and natural language query, simultaneously. Our experimental study has shown that the proposed approach is effective and outperforms the state-of-the-art approaches.

In our future work, we plan to conduct comprehensive experiments on other dataset of different language such as Java or Python, as well as human evaluation to further verify the effectiveness of our proposed approach. Furthermore, we believe that it's promising to explore the potentiality of multi-modal code representation on some other software engineering tasks such as code summarization and code clone detection.

Acknowledgment

This paper is partially supported by the Subject of the Major Commissioned Project "Research on China's Image in the Big Data" of Zhejiang Province's Social Science Planning Advantage Discipline "Evaluation and Research on the Present Situation of China's Image" No. 16YSXK01ZD-2YB, the Zhejiang University Education Foundation under grants No. K18-511120-004, No. K17-511120-017, and No. K17-518051-021, the Major Scientific Project of Zhejiang Lab under grant No. 2018DG0ZX01, the National Natural Science Foundation of China under grant No. 61672453, the Key Laboratory of Medical Neurobiology of Zhejiang Province, and the Foundation of State Key Laboratory of Cognitive Intelligence (Grant No. COGOSC-20190002), iFLYTEK, P.R. China. This work is also supported in part by NSF under grants III-1526499, III-1763325, III-1909323, SaTC-1930941, CNS-1626432, Australian Research Council Linkage Project under LP170100891, and Australian Research Grant DE170101081.

REFERENCES

[1] "GitHub," https://www.github.com, 2019, [Online; accessed 1-May-2019].

[2] "StackOverflow," https://www.stackoverflow.com, 2019, [Online; accessed 1-May-2019].

[3] S. P. Reiss, "Semantics-based code search," in *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 243–253.

[4] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via wordnet for effective code search," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 545–549.

[5] F. Lv, H. Zhang, J.-g. Lou, S. Wang, D. Zhang, and J. Zhao, "Codehow: Effective code search based on api understanding and extended boolean model (e)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 260–270.

[6] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 933–944.

[7] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2016, pp. 87–98.

[8] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, "Improving automatic source code summarization via deep reinforcement learning," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 397–407.

[9] A. V. Aho, R. Sethi, and J. D. Ullman, "Compilers, principles, techniques," *Addison Wesley*, vol. 7, no. 8, p. 9, 1986.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[12] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[13] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing." in *AAAI*, vol. 2, no. 3, 2016, p. 4.

[14] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. Guibas, "Learning program embeddings to propagate feedback on student code," *arXiv preprint arXiv:1505.05969*, 2015.

[15] E. Parisotto, A.-r. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli, "Neuro-symbolic program synthesis," *arXiv preprint arXiv:1611.01855*, 2016.

[16] C. Maddison and D. Tarlow, "Structured generative models of natural source code," in *International Conference on Machine Learning*, 2014, pp. 649–657.

[17] H. K. Dam, T. Tran, and T. Pham, "A deep language model for software code," *arXiv preprint arXiv:1608.02715*, 2016.

[18] W. Ling, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, A. Senior, F. Wang, and P. Blunsom, "Latent predictor networks for code generation," *arXiv preprint arXiv:1603.06744*, 2016.

[19] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, "Bimodal modelling of source code and natural language," in *International Conference on Machine Learning*, 2015, pp. 2123–2132.

[20] J. Chen and H. Zhuge, "Abstractive text-image summarization using multi-modal attentional hierarchical rnn," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 4046–4056.

[21] J.-G. Zhang, P. Zou, Z. Li, Y. Wan, X. Pan, Y. Gong, and P. S. Yu, "Multi-modal generative adversarial network for short product title generation in mobile e-commerce," *NAACL HLT 2019*, pp. 64–72, 2019.

[22] K.-M. Kim, S.-H. Choi, J.-H. Kim, and B.-T. Zhang, "Multimodal dual attention memory for video story question answering," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 673–688.

[23] C. Hori, H. Alamri, J. Wang, G. Wichern, T. Hori, A. Cherian, T. K. Marks, V. Cartillier, R. G. Lopes, A. Das *et al.*, "End-to-end audio visual scene-aware dialog using multimodal attention-based video features," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2352–2356.

[24] M. Carvalho, R. Cadne, D. Picard, L. Soulier, N. Thome, and M. Cord, "Cross-modal retrieval in the cooking context: Learning semantic text-image embeddings," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 2018, pp. 35–44.

[25] L. Ma, Z. Lu, L. Shang, and H. Li, "Multimodal convolutional neural networks for matching image and sentence," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2623–2631.

[26] Y. Cao, M. Long, J. Wang, Q. Yang, and P. S. Yu, "Deep visual-semantic hashing for cross-modal retrieval," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1445–1454.

[27] Q.-Y. Jiang and W.-J. Li, "Deep cross-modal hashing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3232–3240.

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[29] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, "Image captioning with semantic attention," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4651–4659.

[30] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, "The application of two-level attention models in deep convolutional neural network for fine-grained image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 842–850.

[31] J. Lu, J. Yang, D. Batra, and D. Parikh, "Hierarchical question-image co-attention for visual question answering," in *Advances In Neural Information Processing Systems*, 2016, pp. 289–297.

[32] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[33] M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber, "Deep networks with internal selective attention through feedback connections," in *Advances in neural information processing systems*, 2014, pp. 3545–3553.

[34] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.

[35] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[36] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 21–29.

[37] C. Xiong, S. Merity, and R. Socher, "Dynamic memory networks for visual and textual question answering," in *International conference on machine learning*, 2016, pp. 2397–2406.

[38] K. J. Shih, S. Singh, and D. Hoiem, "Where to look: Focus regions for visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4613–4621.

[39] J. Li, M.-T. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *arXiv preprint arXiv:1506.01057*, 2015.

[40] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[41] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," in *International conference on machine learning*, 2016, pp. 1378–1387.

[42] H. Nam, J.-W. Ha, and J. Kim, "Dual attention networks for multimodal reasoning and matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 299–307.

[43] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *arXiv preprint arXiv:1705.04304*, 2017.

[44] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018.

[45] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 423–443, 2019.

[46] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
[48] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *arXiv preprint arXiv:1601.06733*, 2016.
[49] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," *arXiv preprint arXiv:1606.01933*, 2016.
[50] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.
[51] R. Kiros, R. Salakhutdinov, and R. S. Zemel, "Unifying visual-semantic embeddings with multimodal neural language models," *arXiv preprint arXiv:1411.2539*, 2014.
[52] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
[53] Y. Sui and J. Xue, "Svf: interprocedural static value-flow analysis in llvm," in *Proceedings of the 25th international conference on compiler construction*. ACM, 2016, pp. 265–266.
[54] Y. Sui, D. Ye, and J. Xue, "Detecting memory leaks statically with full-sparse value-flow analysis," *IEEE Transactions on Software Engineering*, vol. 40, no. 2, pp. 107–122, 2014.
[55] Y. Sui and J. Xue, "On-demand strong update analysis via value-flow refinement," in *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 2016, pp. 460–473.
[56] M. Kilickaya, A. Erdem, N. Ikizler-Cinbis, and E. Erdem, "Re-evaluating automatic metrics for image captioning," *arXiv preprint arXiv:1612.07600*, 2016.
[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.