2024 IEEE/ACIS 22nd International Conference on Software Engineering Research, Management and Applications (SERA) | 979-8-3503-9134-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SERA61261.2024.10685571

Global Alignment Learning For Code Search

Juntong Hong
Kyoto Institute of Technology
Kyoto, Japan
j-hong@se.is.kit.ac.jp

Eunjong Choi
Kyoto Institute of Technology
Kyoto, Japan
echoi@kit.ac.jp

Kinari Nishiura
Kyoto Institute of Technology
Kyoto, Japan
k-nishiura@kit.ac.jp

Osamu Mizuno
Kyoto Institute of Technology
Kyoto, Japan
o-mizuno@kit.ac.jp

***Abstract*—Code search plays a role in bridging code and query. However, recent code search studies mainly rely on affinity-matrix-based cross-modal attention to learn the word alignments between code and query, which may lead to incorrect alignments. In this paper, we propose a Global Alignment Learning Model (GALM) to learn global alignments and demonstrate that better-learned correct alignments can significantly improve code search performance. Specifically, GALM characterizes the query and code embedding into an alignment graph to enhance the feature representation and further learns global alignments by a dense graph convolutional network. To evaluate the performance of GALM, we compared it with several baseline models on two popular datasets. The results demonstrate that GALM outperforms the best baseline models by 9.8% and 6.8% with the Top@1 accuracy of 0.601 and 0.671 on two datasets, respectively.**

***Index Terms*—code search, deep learning, graph neural network**

I. Introduction

Code search is an essential task in software engineering that aims to retrieve the most relevant code for a given query. Learning correct word alignments between code and query is the key to improving code search performance, similar to automatic code generation [1–3] and code summarization [4–6], while ccurately retrieving relevant code can provide valuable inspiration for writing and reusing code, thus improving software development efficiency.

Recently, code search has benefited much performance from deep-learning technology. Deep-learning-based code search can accurately retrieve the most relevant code and effectively handle unseen tokens by comparing code and query embedding. However, recent deep learning-based code search models face a challenge in using affinity-matrix-based cross-modal attention, as most of these studies [7–12] rely on it to facilitate the model's learning of the alignment between code and query. The affinity-matrix-based cross-modal attention can direct attention between code and query by utilizing the affinity matrix. It comprises crucial word alignment information for word alignment learning and bridges the gap between code and query. On the other hand, it may be unable to learn the differences between the relevant code and similar codes since they might differ only by a few words. Specifically, the affinity-matrix-based cross-modal attention utilizes the max-pooling operation to transform the affinity matrix into the attention weights, which can capture the most salient alignments. However, it may also discard other word alignments and cause the model to learn incorrect alignments since the low co-occurrence frequency of crucial words may be swamped by irrelevant words exhibiting a high co-occurrence frequency.

To address the problem of that incorrect alignment caused by the affinity-matrix-based cross-modal attention, we propose a novel code search model, namely the Global Alignment Learning Model (GALM), which can learn better global alignments. Specifically, GALM first characterizes code and query embeddings into an alignment graph through the cross-modal attention and computes multiblock similarity. Then, instead of using the affinity-matrix-based cross-modal attention, GALM applies a dense graph neural networks (DGCN) on the alignment graph to further capture global alignments. Finally, GALM integrates the global alignments to yield similarity between code and query.

In this paper, based on dot-product attention and code fields (Explained in Section II-A) used in the previous studies [11, 12], we further propose GALM with following contributions:

- GALM characterizes word embedding as an alignment graph to efficiently represent the word alignments and difference information between code and query.
- GALM utilizes dense graph neural networks to further learn global word alignments instead of relying on affinity-matrix-based cross-modal attention.
- We evaluate GALM by comparing it with baseline models. The results demonstrate that GALM outperforms the best baseline models on two datasets.

This paper is organized as follows: Section II describes the background of the technology used in GALM. Section III presents the details of each module of GALM. Section IV evaluates the performance of GALM. Section V summarizes the work presented in this paper.

II. Background

A. Code Fields

In previous studies [11, 12], code tokens are categorized into different code fields instead of plain code. Each code field is transformed into word embeddings by different embedding layers and then concatenated to obtain a distinct code embedding. For example, FcarCS, a code search model proposed by Deng *et al.* [12], it categorizes code tokens into four code fields, method name C_{name} (a list of tokens that split follows camel case), tokens C_{token} (a bag of tokens in the method body), API sequence C_{api} (a list of APIs in method body), and AST fields C_{ast} (a collection of a combination of AST nodes and content). These four code fields are transformed by four different embedding layers and concatenated into a code embedding as follows:

$$C = CoAtt(E(C_{name}), query) \| CoAtt(E(C_{token}), query)$$
$$\| CoAtt(E(C_{api}), query) \| CoAtt(E(C_{ast}), query) \quad (1)$$

where C denotes the code embedding, $CoAtt()$ means the attention mechanism, $E()$ means word embedding layer, and $\|$ denotes concatenate operation.

Similarly, the query Q is also transformed by an embedding layer into a word embedding as follows:

$$Q = E(Q). \quad (2)$$

where the Q denotes the query embedding, $E()$ denotes word embedding layer.

B. Attention Mechanism

The attention mechanism's role in the code search is to align words between code and query cross-modally. It can usually operate in two different approaches.

The first one is to utilize the affinity matrix with the pooling operation, which can be formulated as follows:

$$\begin{aligned} &\text{Attention}_c = \text{maxpooling}(\text{Affinity}(c, q)), \\ &\text{Attention}_q = \text{maxpooling}(\text{Affinity}(c, q)^\top), \\ &\text{Affinity}(c, q) = WQC^\top \end{aligned} \quad (3)$$

where the $\text{Attention}_c \in \mathbb{R}^{m\times m}$ and $\text{Attention}_q \in \mathbb{R}^{n\times n}$ denote the attention weights for code C and query Q, which are transformed from the affinity matrix $\text{Affinity}(c, q) \in \mathbb{R}^{n\times m}$. Here, maxpooling() means the max-pooling operation, $\top$ means the transpose operation, and W is the learnable variable. The Q and C denote code and query embedding, respectively. This approach involves utilizing the pooling operation to transform the attention weights. It can filter out irrelevant word alignments by discarding low-score alignments. However, it may also incorrectly discard other essential word alignments.

Another approach is using dot-product attention [13], which can retain whole alignment information. It can be formulated as follows:

$$\begin{aligned} &a = Softmax(QC^\top) \\ &C_{weighted} = Q^\top a \end{aligned} \quad (4)$$

where $a \in \mathbb{R}^{n\times m}$ denotes the attention weights, m and n denote the number of words of code and query, respectively. The $C_{weighted} \in \mathbb{R}^{m\times dim}$ refers the weighted code, $\top$ denotes transpose operation. $C \in \mathbb{R}^{m\times dim}$ and $Q \in \mathbb{R}^{n\times dim}$ means the query embedding and code embedding.

III. Proposed Model

The overview of our proposed model, GALM, is shown in Fig. 1. To address the incorrect alignment caused by affinity-matrix-based cross-modal attention, GALM computes global alignment instead. GALM consists of the following three modules:

- **Word embedding module** transforms the inputted unified code field and query into code and query embedding, as described in Section III-A.
- **Alignment graph construction module** characterizes the alignments between code and query as an alignment graph, as described in Section III-B.
- **Graph similarity measuring module** further learns the global alignment representations from the alignment graph and integrates it into the global similarity, as described in Section III-C.

In this section, we first explain the each module of GALM and then introduce the objective function in Section III-D.

A. Word Embedding Module

This module transforms inputted code fields and query into high-dimensional word embeddings. To this end, GALM employs three code fields (method name, token, API), which differs from previous studies [11, 12] using four code fields (method name, token, API, AST). (Code fields are explained in Section II-A) GALM concatenates these code fields to create the unified code field and then transforms it into a code embedding using an embedding layer. The embedding procedure for code and query can be formulated as follows:

$$\begin{aligned} &C = E(C_{name} \| C_{token} \| C_{api}), \\ &Q = E(Q_{query}) \end{aligned} \quad (5)$$

where the code embedding C is set of word embeddings $C = \{c_1, ..., c_m\}, C \in \mathbb{R}^{m\times dim}$, m means the number of the word embedding, $\|$ denotes the concatenate operation.

Similarly, query embedding is also transformed by a query embedding layer.

$$Q = E(Q_{query}) \quad (6)$$

The query embedding Q is also a set of word embeddings $Q = \{q_1, ..., q_n\}, Q \in \mathbb{R}^{n\times dim}$, n means the number of the word embedding. The dim means the word embedding dimension.

B. Alignment Graph Construction Module

The objective of this module is to learn cross-modal word alignments between code and query and represent these alignments as an alignment graph.

Cross-Modal Alignment: Cross-modal alignment submodule plays a pivotal role in bridging the code and query at the initial stage. It takes code embedding C and query embedding Q as input and computes dot-product attention (Explained in Section II-B) between them to acquire cross-modal weights in cross-modal alignment submodule. These weights are applied to the query to obtain the code-weighted query. The procedure can be formulated as follows:

$$\hat{Q} = Softmax(QC^\top)C \quad (7)$$

where the $\hat{Q} \in \mathbb{R}^{n\times d}$ represents the code-weighted query, $\top$ denotes transpose operation.

Diverse Alignment Representation: This submodule further enriches code-weighted query as diverse alignment representation with the difference information between the

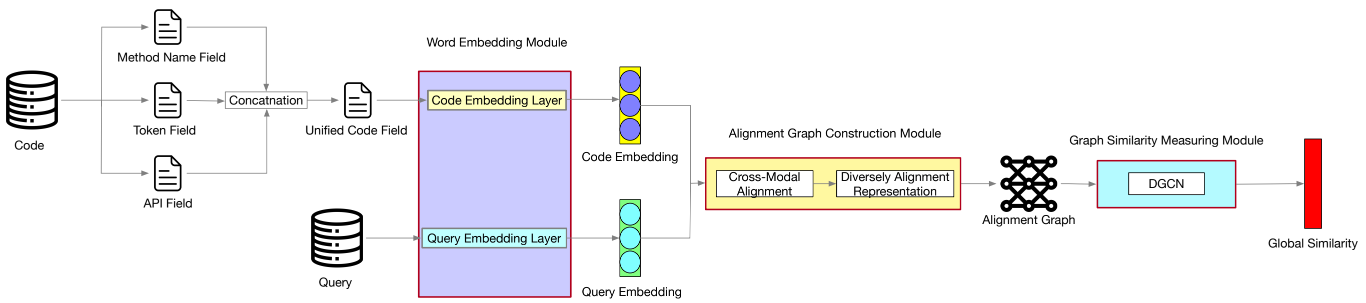

Fig. 1: The overview of GALM, given code and query pair, GALM transforms them to word embeddings in the word embedding module and then characterizes them as an alignment graph in the alignment graph construct module. Finally, the graph similarity measuring module calculates and integrates the global alignments into global similarity.

code-weighted query and query embedding since the alignment information in the code-weighted query is insufficient to fully express the difference for computing the similarity to relevant code. Therefore, the diverse alignment representation can express the difference between the code-weighted query and query embedding while incorporating the code-query alignments.

To this end, similar to Vaswani *et al.* [14] and Liu *et al.* [15], the code-weighted query $\hat{Q}$ and query embedding Q are first divided into the multiblock representation to enhance the feature representation. The dividing procedure can be formulated as follows:

$$\begin{aligned} \hat{Q}^G &= \hat{Q}_1 \| \hat{Q}_2 \dots \hat{Q}_i, \\ Q^G &= Q_1 \| Q_2 \dots Q_i \end{aligned} \tag{8}$$

where the $\hat{Q}^G \in \mathbb{R}^{n \times i \times k}$ and $Q^G \in \mathbb{R}^{n \times i \times k}$ denote the multiblock representation of code-weighted query $\hat{Q}$ and query embedding Q, composed of concatenation by $\hat{Q}_i \in \mathbb{R}^{n \times 1 \times k}$ and $Q_i \in \mathbb{R}^{n \times 1 \times k}$, which the i-th blocks of $\hat{Q}^G$ and Q^G, each block has the dimension of k, and $\|$ denotes the concatenation operation.

Subsequently, this submodule calculates words cosine similarity between of the multiblock representation of code-weighted query $\hat{Q}$ and query embedding Q to include the difference information into the code-weighted query. The procedure can be formulated below:

$$V = Cosine(\hat{Q}^G, Q^G) \tag{9}$$

where the the $V \in \mathbb{R}^{n \times k}$ denotes diverse alignment representation, the $Cosine()$ denotes the cosine similarity function. Note that the diverse alignment representation V is then treated as the alignments graph with n nodes, where each node has k dimensions. Within this graph, each query word is treated as a node, and each node is densely connected.

C. Graph Similarity Measuring Module

This module takes alignment graph V as input. It utilizes a Dense Graph Convolutional Network (DGCN), a variant of a graph convolutional network (GCN) [16, 17], to learn global alignments from alignment graph V and integrate them into global similarity.

The initial edge weights of DGCN can be computed as follows:

$$E_q = Softmax(Q^\top Q) \tag{10}$$

where the $E_q \in \mathbb{R}^{n \times n}$ denotes the initial edge weights for DGCN. The $\top$ means the transpose operation.

Dense Graph Convolutional Network (DGCN): DGCN updates the edge weights as follows:

$$E_D = Softmax(\hat{Q}^{d\top} \hat{Q}^d) \tag{11}$$

where the $E_D \in \mathbb{R}^{m \times m}$ represents the updated edge weights, which updated by the current global alignments $\hat{Q}^d$, the $\top$ denotes the transpose operation. Then, DGCN utilizes the edge weights E_D to update the alignments graph V to obtain the global alignments. The process can be formulated as follows:

$$\hat{Q}^d = \prod_{l=1}^{L} Relu(V E_D W_d + b_D) \tag{12}$$

where $\hat{Q}^d \in \mathbb{R}^{n \times dim}$ denote the global alignments, the l represents the l-th layer and the L denotes the number of iterates DGCN. The $W_d \in \mathbb{R}^{dim \times dim}$ represents the trainable parameter and $b_D \in \mathbb{R}^{dim}$ denotes the bias.

To integrate the global alignment $\hat{Q}^d$ to global similarity $\hat{Q}_{df}$, DGCN employs a two-layer MLP and applies global average pooling. The following illustrates the process:

$$\text{similarity} = \frac{1}{n} W_{do} \tanh(\hat{Q}^d W_{di} + b_{di}) + b_{so} \tag{13}$$

where the similarity represents the global similarity, which the similar between code and query. The n denotes the number of nodes. The $W_{do} \in \mathbb{R}^{dim \times dim}$ and $W_{di} \in \mathbb{R}^{dim \times 1}$ denote the trainable parameters, and the $b_{di} \in \mathbb{R}^{dim}$ and $b_{so} \in \mathbb{R}^{dim}$

D. Objective Function

We use Bi-directional ranking loss [18–20] as the objective function to optimize GALM. The optimization goal is to have the query closer to relevant code than irrelevant code. Bi-directional ranking loss can be formulated as follows:

$$\begin{aligned} loss(C, Q) = \max[0, \lambda - Sim(c, q) + Sim(c, \hat{q})] + \\ [0, \lambda - Sim(c, q) + Sim(\hat{c}, q)] \end{aligned} \tag{14}$$

where the λ denotes the margin, the $Sim()$ refers to the similarity function. Moreover, c, q denotes the relevant code and query pair, $c, \hat{q}$ denotes code and irrelevant query pair, and $\hat{c}, q$ denotes the irrelevant code and query pair.

IV. Experiments

A. Evaluation Methodology

1) Dataset: We evaluate GALM on two popular java datasets: **CSN dataset** and **JCG dataset**. CSN dataset, originally provided by CodeSearchNet [22] and preprocessed by Xu *et al.* [11][1], is a multifield code search dataset containing 394,497 training code and query pairs. The **JCG dataset**, originally provided by Hu *et al.* [5] and subsequently preprocessed by Deng *et al.* [12][2], is another multifield code search dataset containing 475,463 training code and query pairs. Both datasets include 10,000 test code and query pairs, each comprising a query field and four code fields: method name, token, API and AST field. Note that this study uses the data splits in the publicly accessible datasets. Furthermore, we removed 30 code and query pairs from the test datasets because several pairs had blank fields.

2) Parameters: We have set the hidden dimension to 256 and configured the alignment graph with 16 blocks, where each block having a hidden dimension of 16. We employed Adam as the optimizer, and the initial learning rate is set to 0.0003, which is reduced to 0.00005 after 15 epochs, iterating in 30 epochs.

3) Measurements: We evaluate GALM using two metrics: namely **Top@K** and Mean Reciprocal Rank (**MRR**), two commonly used metrics for code search research.

Top@K can be formulated as follows:

$$Top@K = \frac{1}{Q}\sum_{q=1}^{Q}\begin{cases}1 & Rank_\sigma <= K\\ 0 & Rank_\sigma > K\end{cases} \tag{15}$$

where K signifies whether the codes retrieved of rank K contain the most relevant code, Q is the total number of code and query pairs, $Rank_\sigma$ denotes the rank of retrieved code.

MRR is a metric that calculating average of the reciprocal ranks of results for a set of queries. MRR can be formulated as follows:

$$MRR = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{Rank_\sigma} \tag{16}$$

where Q is the total number of code and query pairs, $Rank_\sigma$ means the rank of retrieved code.

B. Comparison Experiments

To evaluate the code search performance of GALM, we conduct competitive experiments with baseline models. We referenced the results from TabCS and FcarCS on the CSN and JCG datasets, respectively.

[1]https://github.com/cqu-isse/TabCS

[2]https://github.com/cqu-isse/FcarCS

TABLE I: Result on CSN dataset

Models	Top@1	Top@5	Top@10	MRR
DeepCS	0.294	0.440	0.529	0.307
CARLCS-CNN	0.413	0.557	0.633	0.409
CARLCS-TS	0.439	0.580	0.656	0.426
UNIF	0.42	0.556	0.624	0.419
TabCS w/o AST	0.508	0.654	0.724	0.504
TabCS	0.547	0.683	0.748	0.539
GALM (Ours)	**0.601**	**0.781**	**0.821**	**0.677**

TABLE II: Result on JCG dataset

	Top@1	Top@5	Top@10	MRR
DeepCS	0.278	0.434	0.516	0.282
UNIF	0.529	0.682	0.772	0.525
TabCS	0.550	0.719	0.792	0.544
FcarCS w/o AST	0.609	0.757	0.823	0.593
FcarCS	0.628	0.770	0.830	0.613
GALM (Ours)	**0.671**	**0.853**	**0.894**	**0.751**

1) Baseline Models: On CSN dataset, we compared GALM with DeepCS, CARLCS-TS, UNIF [21], CARLCS-CNN and TabCS. On JCG dataset, we compared GALM with DeepCS, UNIF, TabCS, and FcarCS. In addition, we further compared GALM with the models using same limited code fields (i.e., method name, token, API), namely TabCS w/o AST and FcarCS w/o AST. However, TabCS w/o AST is specifically for the CSN dataset only, FcarCS w/o AST is specifically for JCG dataset only.

2) Comparison Results: **Comparison results on CSN dataset.** Table I shows the results of compared GALM with baseline models. As can be seen, GALM achieves an MRR of 0.677 and Top@1, Top@5, and Top@10 scores of 0.601, 0.781, and 0.821, respectively. GALM outperforms TabCS, by 25.6%, 9.8%, 14.3%, and 9.7% in terms of MRR and Top@1/5/10, respectively. These results illustrate that GALM outperforms the best baseline model TabCS with limited code fields, and the GALM's features, the diverse alignment representation, and global alignments can significantly enhance code search accuracy.

Comparison results on JCG dataset. Table II illustrates the results of GALM on the JCG dataset compared with baseline models. As can be observed, GALM achieves an MRR of 0.777 and Top@1/5/10 scores of 0.702, 0.873, and 0.908, respectively. GALM outperforms the FcarCS by 26.7%, 6.8%, 10.7%, and 7.7% in MRR and Top@1/5/10. The experiments on the JCG dataset show GALM demonstrated higher performance than the best baseline model FcarCS with limited code fields and GALM performed consistently on different datasets.

Comparison results using the same code fields. The results that GALM compared with TabCS w/o AST and FcarCS w/o AST are also shown in both Table I and II, respectively. As can be seen, GALM outperforms TabCS w/o AST in MRR by 34.3%, and Top@1/5/10 by 18.3%, 19.3%, and 13.3% on CSN dataset, respectively. Compared to FcarCS w/o AST, GALM outperforms by 26.6% in MRR and by 12.6%, 12.6%, and 8.6% in terms of Top@1/5/10 on JCG dataset, respectively. GALM show a wider margin when compared to baseline models that use the same number of code fields.

These results demonstrate our model architecture's effectiveness, and better learned alignments significantly improved model performance. Moreover, these results show that the advantage of a well-designed model architecture can bring more significant potential for improved performance than the use of additional inputs.

V. Conclusion

In this paper, we propose a novel code search model, namely GALM. GALM characterizes the code and query embedding as an alignment graph to enhance the code distinction and learn better alignments. It further utilizes dense graph convolutional networks for alignments learning to address the issue of incorrect alignments caused by affinity-matrix-based cross-modal attention. To evaluate the performance of GALM, we conduct comparison experiments with several baseline models on two datasets. The results show that GALM outperforms best baseline models by a large margin.

References

[1] R. Zhong, M. Stern, and D. Klein, "Semantic scaffolds for pseudocode-to-code generation," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2283–2295.

[2] P. Yin and G. Neubig, "TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, pp. 7–12.

[3] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, and L. Zhang, "Treegen: A tree-based transformer architecture for code generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[4] A. LeClair, S. Haque, L. Wu, and C. McMillan, "Improved code summarization via a graph neural network," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, p. 184–195.

[5] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th Conference on Program Comprehension (ICPC)*, 2018, p. 200–210.

[6] C. Chen, X. Peng, Z. Xing, J. Sun, X. Wang, Y. Zhao, and W. Zhao, "Holistic combination of structural and textual code information for context based api recommendation," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2987–3009, 2022.

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

[8] J. Shuai, L. Xu, C. Liu, M. Yan, X. Xia, and Y. Lei, "Improving code search with co-attentive representation learning," in *Proceedings of the 28th International Conference on Program Comprehension (ICPC)*, 2020, pp. 196–207.

[9] Q. Huang, A. Qiu, M. Zhong, and Y. Wang, "A code-description representation learning model based on attention," in *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 447–455.

[10] Y. Cheng and L. Kuang, "Csrs: Code search with relevance matching and semantic matching," in *Proceedings of the 26th Conference on Program Comprehension (ICPC)*, pp. 533–542, 2022.

[11] L. Xu, H. Yang, C. Liu, J. Shuai, M. Yan, Y. Lei, and Z. Xu, "Two-stage attention-based model for code search with textual and structural features," in *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 342–353.

[12] Z. Deng, L. Xu, C. Liu, M. Yan, Z. Xu, and Y. Lei, "Fine-grained co-attentive representation learning for semantic code search," in *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 396–407.

[13] Thang. L, Hieu. P, and Christopher D., "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] C. Liu, Z. Mao, T. Zhang, H. Xie, B. Wang, and Y. Zhang, "Graph structured network for image-text matching," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 921–10 930.

[16] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Proceedings of the convolutional networks on graphs for learning molecular fingerprints," in *Neural Information Processing Systems*, 2015.

[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2016.

[18] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma *et al.*, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International journal of computer vision*, vol. 123, no. 1, 2017.

[19] K.-H. Lee, X. Chen, G. Hua, H. Hu, and X. He, "Stacked cross attention for image-text matching," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 201–216.

[20] F. Faghri, D. J. Fleet, J. R. Kiros, and S. Fidler, "Vse++: Improving visual-semantic embeddings with hard negatives," in *Proceedings of The British Machine Vision

Association, 2017.

[21] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, “When deep learning met code search,” in *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 964–974.

[22] H. Husain, H.H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, “Codesearchnet challenge: Evaluating the state of semantic code search,” Computer Research Repository, vol.abs/1909.09436, 2019.