Hyperbolic Code Retrieval: A Novel Approach for Efficient Code Search Using Hyperbolic Space Embeddings

Xunzhu Tang[1]*, Zhenghan Chen[2†], Saad Ezzini[3], Haoye Tian[1], Yewei Song[1], Jacques KLEIN[1], Tegawendé F. Bissyandé[1]

[1]University of Luxembourg, Luxembourg
[2]Peking University, Beijing, China
[3]Lancaster University, Lancaster, UK

xunzhu.tang@uni.lu, 1979282882@pku.edu.cn, s.ezzini@lancaster.ac.uk, haoye.tian@uni.lu, yewei.song@uni.lu, jacques.klein@uni.lu, tegawende.bissyande@uni.lu

Abstract

Within the realm of advanced code retrieval, existing methods have primarily relied on intricate matching and attention-based mechanisms. However, these methods often lead to computational and memory inefficiencies, posing a significant challenge to their real-world applicability. To tackle this challenge, we propose a novel approach, the Hyperbolic Code QA Matching (`HyCoQA`). This approach leverages the unique properties of Hyperbolic space to express connections between code fragments and their corresponding queries, thereby obviating the necessity for intricate interaction layers. The process commences with a reimagining of the code retrieval challenge, framed within a question-answering (QA) matching framework, constructing a dataset with triple matches characterized as <negative code, description, positive code>. These matches are subsequently processed via a static BERT embedding layer, yielding initial embeddings. Thereafter, a hyperbolic embedder transforms these representations into hyperbolic space, calculating distances between the codes and descriptions. The process concludes by implementing a scoring layer on these distances and leveraging hinge loss for model training. Especially, the design of `HyCoQA` inherently facilitates self-organization, allowing for the automatic detection of embedded hierarchical patterns during the learning phase. Experimentally, `HyCoQA` showcases remarkable effectiveness in our evaluations: an average performance improvement of 3.5% to 4% compared to state-of-the-art code retrieval techniques.

Introduction

In the domain of software development, code search has become an essential pursuit for developers. Frequently, they dedicate significant time to combing through existing codebases in search of fragments that align with their needs. The aim of code search is to uncover code snippets within repositories that reflect users' intentions, often articulated in natural language. The proliferation of extensive code libraries, exemplified by platforms like GitHub and StackOverflow, has introduced a formidable challenge: efficiently retrieving semantically equivalent code from a vast array of possibilities (Gu, Zhang, and Kim 2018a; Liu et al. 2021; Di Grazia and Pradel 2023; Tang et al. 2023).

In the past, coding was an isolated pursuit centered on translating logic into machine-readable instructions. Yet, modern development's collaborative landscape champions code reuse and modularity. Efficiently harnessing existing code is now essential, underscoring the need for a sophisticated code search mechanism. Such a mechanism must surpass syntax matching, comprehending the intricate semantics and intent in both code and queries. Early strategies, based on traditional information retrieval, relied on keyword matches (McMillan et al. 2011a; Lv et al. 2015a; Linstead et al. 2009), lacking the nuance to untangle deep semantics or fathom natural language subtleties, often yielding suboptimal outcomes. The rise of deep learning and natural language processing heralded a transformative phase in code search (Gu, Zhang, and Kim 2018a; Cambronero et al. 2019; Xu et al. 2021; Chai et al. 2022; Shuai et al. 2020; Cheng and Kuang 2022; Sun et al. 2022), shifting towards encoding code and queries into dense semantic spaces to bridge the gap between abstract requirements and tangible implementations. This evolution is punctuated by notable landmarks: Information Retrieval Paradigms: Early code retrieval leaned on conventional methods, transforming queries and code using algorithms like TF-IDF (McMillan et al. 2011a; Lv et al. 2015a; Linstead et al. 2009). Deep Learning Inroads: The resurgence of neural networks and deep learning shifted the landscape, encoding code and queries into dense vectors to grasp semantic subtleties. Pre-trained Model Epoch: Recent research embraced pre-trained models such as CodeBERT, CodeRetriever, and CoCoSoDa (Feng et al. 2020a; Li et al. 2022a; Shi et al. 2023a), harnessing extensive datasets and intricate training to bridge the divide between natural language and code.

While each research avenue has indubitably advanced the domain, the aspiration for an optimal code retrieval system remains unfulfilled. Each method, while groundbreaking in its own right, encapsulates inherent limitations. Furthermore, the intricate interplay between natural language descriptions and code is rife with latent relationships and differences that extant methodologies might not fully capture.

The realm of mathematics frequently unveils insights and tools that hold potential for addressing intricate challenges in diverse fields. In the context of code retrieval, one such mathematical concept, hyperbolic geometry, emerges as a

*Corresponding author
†These authors contributed equally.
Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

promising contender. Unlike traditional Euclidean spaces, hyperbolic spaces excel at depicting hierarchical structures, which often underlie the relationship between code and its corresponding natural language description. In this context, our study poses a fundamental question: Can the distinctive attributes of hyperbolic spaces be harnessed to construct a more potent and semantically conscious code retrieval system? In this paper, we embark on a journey to address this inquiry. We introduce "Hyperbolic Code QA Matching" or HyCoQA, a pioneering approach that reimagines the very essence of code retrieval. By seamlessly integrating the inherent qualities of hyperbolic spaces with cutting-edge embedding techniques, we aspire to encapsulate the intrinsic hierarchies and relationships inherent in the code-description interplay. We contend that this methodology, underpinned by the mathematical precision of hyperbolic geometry, has the potential to bridge the semantic gap between natural language queries and code with unparalleled efficiency.

Our contributions can be summarized as follows:

- **Novel Hyperbolic Architecture**: We propose "Hyperbolic Code QA Matching" (HyCoQA), an innovative approach that leverages Hyperbolic space to establish relationships between code fragments and queries, simplifying complex interaction mechanisms.
- **QA Framework Redefinition**: We redefine code retrieval as a question-answering framework, processing triple matches through a static BERT embedding layer to create initial embeddings.
- **Enhanced Efficacy**: Compared to existing solutions, HyCoQA showcases superior performance, achieving 3.5%-4% average improvement against leading code retrieval methods.

Approach

As illustrated in Figure 1, each description is paired with a correct code segment (positive answer) and an erroneous one (negative answer). Drawing inspiration from the natural language processing domain, we posit that the relationship between a description and its associated code mirrors the question-answer (QA) dynamic prevalent in NLP. Specifically, while the description elucidates the problem, the code delineates the solution to that problem. In this context, our primary objective is to maximize the margin between the scores of the correct QA pair and the negative QA pair, ensuring that the system can robustly differentiate between accurate and inaccurate solutions based on the given description.

Transformation: From Code Retrieval to QA Pair Matching In the code retrieval process, given a description, the objective is to identify and validate the presence of any pertinent code within the top N retrieved codes. To streamline this, we transform the code retrieval task into a QA pair matching paradigm: for a given description, we pair it with an accurate code (designated as positive) and a randomly selected inaccurate code (designated as negative). The primary training objective is to optimize the model to widen the gap between the scores of the accurate QA pair and the erroneous QA pair. During the testing phase, our refined hyperbolic model is employed to embed both codes and descriptions. Subsequently, for a presented description, the system evaluates the presence of the appropriate code among the top N retrieved codes.

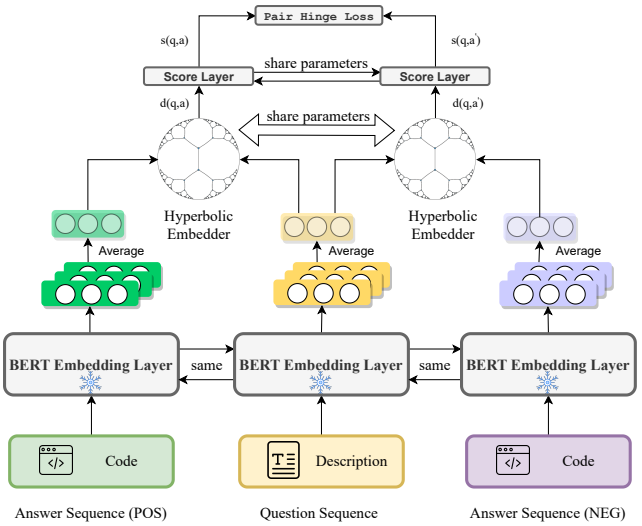

Figure 1: Architecture of HyCoQA.

BERT Embedding Layer

To adeptly comprehend the relationship between descriptions and code, it's imperative to translate textual sequences into their corresponding numerical representations. Our architecture processes three distinct sequences: the question (denoted as q), the accurate answer (symbolized as a), and a randomly chosen incorrect answer (referred to as a'). Each of these sequences contains M words, where M_q and M_a represent the predetermined maximum sequence lengths for questions and answers, respectively.

While numerous methodologies (Bafna, Pramod, and Vaidya 2016; Pennington, Socher, and Manning 2014; Church 2017; Joulin et al. 2016) exist for the transformation of text into vector representations, the optimal selection of an embedding technique is paramount. This choice directly impacts the fidelity with which the vectors capture textual nuances. Contrary to widely-adopted models like Word2Vec (Mikolov et al. 2013), which assigns a static representation to each word irrespective of its surrounding context, BERT (Devlin et al. 2019) offers a more nuanced approach. Specifically, BERT yields word vectors that dynamically adjust based on the context provided by adjacent words while in our work, we choose static BERT.

Owing to these merits, we employ BERT (Devlin et al. 2019) as our foundational embedding model for both descriptions and tokenized code segments. Furthermore, in order to make our model's model faster, the parameters of BERT are frozen. The iteration we utilize is a pre-trained expansive model, comprising 24 layers and an embedding dimensionality of 1,024, fine-tuned on cased English literature. Once the text sequences are embedded in this vector space, it enables us to execute various numerical operations on them, such as determining textual similarity or computing correlation metrics.

Task-Specific Word Representation via Projection

To derive a word representation tailored to our task, we employ a projection layer. This layer is conceptualized as a singular neural network layer, impacting each word present in the three sequences.

$$\begin{aligned} x &= \sigma\left(\mathbf{W}_p z + b_p\right) \\ \text{subject to} \quad & \mathbf{W}_p \in \mathbb{R}^{d \times n}, \quad z \in \mathbb{R}^n, \quad x \in \mathbb{R}^d \end{aligned} \tag{1}$$

where σ is a non-linear function, such as the rectified linear unit (ReLU). The outcome of this layer consists of a set of d-dimensional embeddings corresponding to each sequence, namely the question, the positive answer, and the negative answer. Crucially, the parameters intrinsic to this projection layer are consistently shared across the question and its associated answer.

Deriving QA Representations

To extract representations for questions and answers, we straightforwardly aggregate all word embeddings within the sequence.

$$y^* = \sum_{i=1}^{M_*} x_i^* \tag{2}$$

In this equation, $*$ encompasses $\{q, a, a'\}$. M denotes the preset maximum sequence length (pertinent to both question and answer), while $x_1, x_2, \ldots, x_M$ are the d-dimensional sequence embeddings. Furthermore, we normalize the question and answer embeddings to fit within the unit sphere before progressing to subsequent layers, ensuring $\|y^*\| \leq 1$. This is achieved through $y^* = \frac{y^*}{\|y^*\|}$ whenever $\|y^*\| > 1$. Emphasizing, this normalization of QA embeddings to the unit sphere is imperative for the optimal functionality of `HyCoQA`.

Embedding Interactions within a Hyperbolic Riemannian Framework for QA Pairs

In the realm of neural ranking, the choice of interaction function between representations of questions and answers serves as a defining attribute. Within the scope of our research, we predominantly employ the hyperbolic distance function [1] to elucidate the intricate relationships embedded within questions and answers. Explicitly, let's consider $\mathcal{B}^d$ as the open d-dimensional unit ball, defined as $\left\{x \in \mathbb{R}^d \mid \|x\| < 1\right\}$. Our model is conceptualized within the Riemannian manifold $\left(\mathcal{B}^d, g_x\right)$ and is endowed with a specific Riemannian metric tensor, which can be expressed as:

$$\begin{aligned} g_x &= \left(\frac{2}{1 - \|x\|^2}\right)^2 g^E \\ \textit{s.t.} \quad & g^E \text{ is the Euclidean metric tensor} \end{aligned} \tag{3}$$

Delving into the hyperbolic distance function that characterizes the interaction between the question and answer, it can be delineated as:

$$\begin{aligned} d(q, a) &= \operatorname{arcosh}\left(1 + 2\frac{\|q - a\|^2}{\left(1 - \|q\|^2\right)\left(1 - \|a\|^2\right)}\right) \\ \textit{s.t.} \quad & q, a \in \mathbb{R}^d \end{aligned} \tag{4}$$

The term "arcosh" is synonymous with the inverse hyperbolic cosine function, represented as $\operatorname{arcosh} x = \ln\left(x + \sqrt{x^2 - 1}\right)$. Notably, the value of $d(q, a)$ exhibits a nuanced variation predicated on the spatial positioning of q and a. This fluidity fosters the organic discovery of latent hierarchies. Given this configuration, an exponential surge in distance is observed as the vector's norm approaches unity. This phenomenon results in the encapsulation of inherent hierarchies within the QA embeddings through the vector's norm. From a geometric vantage point, the origin is visualized as a tree's root, proliferating expansively towards the periphery of the hyperbolic ball. The innate ability of the hyperbolic distance to discern hierarchies is elucidated both graphically and qualitatively in subsequent segments.

Gradient Computation Among the various hyperbolic geometric models, the Poincarè hyperbolic distance stands out due to its differentiability. Given this, the partial derivative with respect to θ is:

$$\begin{aligned} \frac{\partial d(\theta, x)}{\partial \theta} &= \frac{4}{\beta\sqrt{\gamma^2 - 1}}\left(\frac{\|x\|^2 - 2\langle\theta, x\rangle + 1}{\alpha^2}\theta - \frac{x}{\alpha}\right) \\ \textit{s.t.} \quad & \alpha = 1 - \|\theta\|^2, \\ & \beta = 1 - \|x\|^2, \\ & \gamma = 1 + \frac{2}{\alpha\beta}\|\theta - x\|^2. \end{aligned} \tag{5}$$

While various hyperbolic geometric models are available, such as the Beltrami-Klein and Hyperboloid models, our preference is the Poincarè ball/disk due to its differentiation simplicity and absence of constraints (Nickel and Kiela 2017).

Hyperbolic Distance-Based Similarity Computation

In the intricate architecture of our model, the hyperbolic distance's transformation through a linear layer forms a pivotal step. This step ensures that the abstract spatial relationships in the hyperbolic space are mapped to values that can be utilized effectively in subsequent layers. The transformation is represented as:

$$\begin{aligned} s(q, a) &= w_f d(q, a) + b_f \\ \textit{s.t.} \quad & w_f \in \mathbb{R}^1, \\ & b_f \in \mathbb{R}^1 \end{aligned} \tag{6}$$

The parameters, w_f and b_f, are scalar components that govern this transformation, adjusting the scale and bias respectively. Their significance is underscored by empirical evidence: this layer has been chosen after a rigorous evaluation process, which considered various alternatives and their performance metrics.

Optimization Techniques and Learning Paradigm

The realm of optimization in neural architectures, especially those operating in non-Euclidean spaces, is vast and intricate. Within the HyCoQA framework, the optimization strategy leans heavily on a pairwise ranking loss, aligning perfectly with the metric-centric nature of the model.

Incorporation of Pairwise Hinge Loss To ensure the model discerns correct answers from incorrect ones effectively, it is trained to minimize a pairwise hinge loss. This loss function is articulated as:

$$L = \sum_{(q,a)\in\Delta_q} \sum_{(q,a')\notin\Delta_q} \max\left(0, s(q,a) + \lambda - s\left(q,a'\right)\right)$$
$$\textit{s.t.} \quad \Delta_q \text{ is the set of all QA pairs for question } q \tag{7}$$

The incorporation of the pairwise hinge loss is not arbitrary; its selection is rooted in empirical results, demonstrating its superior performance in similar scenarios.

Riemannian Optimization Navigating the landscape of hyperbolic space presents unique challenges, especially when it comes to gradient-based optimization. Recognizing this, our model utilizes Riemannian optimization techniques:

$$\theta_{t+1} = \mathfrak{R}_{\theta_t}\left(-\eta \nabla_R \ell\left(\theta_t\right)\right)$$
$$\textit{s.t.} \quad \mathfrak{R}_{\theta_t} \text{ denotes a retraction to } \mathcal{B} \text{ at } \theta \tag{8}$$

The Riemannian gradient has a close relationship with its Euclidean counterpart, which offers computational advantages:

$$\nabla_R = \frac{\left(1 - \|\theta_t\|^2\right)^2}{4} \nabla_E$$
$$\textit{s.t.} \quad \nabla_E \text{ is the Euclidean gradient} \tag{9}$$

Owing to the complexity and nuances of working in hyperbolic space, we steer readers seeking a deeper understanding towards references (Bonnabel 2013; Nickel and Kiela 2017). In the implementation phase, the power of TensorFlow's gradient computation is harnessed, albeit with necessary transformations as detailed above.

Evaluative Retrieval during Testing with HyCoQA

As shown in Figure 2, during the evaluative phase, the proficiency of the trained HyCoQA comes to the fore, allowing for the discernment of well-matched question-answer pairs. Given a description, denoted as d, and assuming the availability of N code snippets for retrieval, embeddings for d and the corresponding N code vectors are derived as per Equation 10.

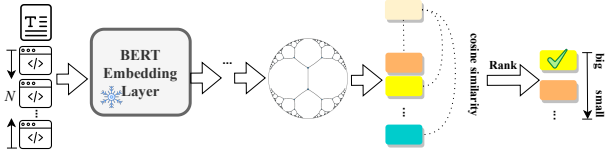

Figure 2: Test Stage

$$C = (\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N)$$
$$\text{s.t.} \quad \mathbf{c}_i = \texttt{HyCoQA}(c_i) \tag{10}$$

Subsequent to obtaining the set C of embedded codes, it is ranked based on their relevance to the description d. The primary evaluative criterion is the position of the ground truth code within this ranked set; specifically, we assess whether the actual code associated with d appears within the top N entries of C.

Experimental Design

In this section, we present our experimental setup, metrics, baselines, and research questions.

Dataset

As depicted in Figure 3, the CodeSearchNet serves as a pivotal benchmark in the domain of code searching. Comprising over 2 million code snippets sourced from GitHub, this dataset spans six distinct programming languages: Go (726,768 snippets), Java (1,569,889 snippets), JavaScript (1,857,835 snippets), PHP (977,821 snippets), Python (1,156,085 snippets), and Ruby (164,048 snippets). The primary objective of CodeSearchNet is to facilitate developers in efficiently locating the requisite code. Furthermore, it catalyzes advancements in research areas such as natural language processing and code search methodologies.

It is imperative to note that the values presented in Figure 3 also denote the quantity of positive pairs. Specifically, the number of snippets with documentation for each language are as follows: Go (347,789), Java (542,991), JavaScript (157,988), PHP (717,313), Python (503,502), and Ruby (57,393). In the context of our research, we employ a stochastic approach to select code not aligned with the ground truth to form a negative pair. Consequently, our dataset structure manifests as <positive code, description, negative code>, maintaining the dimensions elucidated in Figure 3.

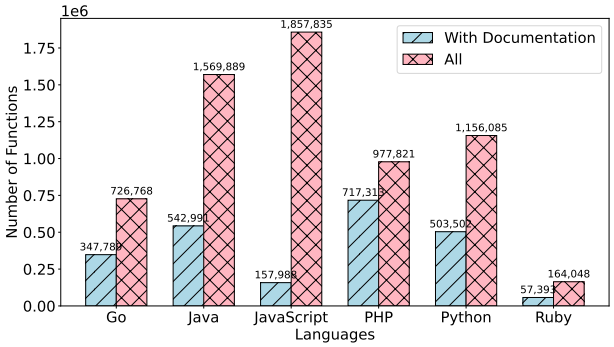

Figure 3: Dataset Size Statistics by Language

Evaluation Metrics

In assessing the effectiveness of our proposed methodology, we adopt a series of metrics, which have been consistently recognized in prevailing research (Gu, Zhang, and Kim 2018b; Du et al. 2021; Wan et al. 2019; He et al. 2020). Specifically, we employ the mean reciprocal rank (MRR) complemented

by top-k recall, represented as $R@k$ where $k \in \{1, 5, 10\}$. The MRR metric furnishes a nuanced evaluation by ascertaining the average of the inverse ranks of the relevant code snippets corresponding to a designated set of queries, Q. In contrast, $R@k$ offers an aggregate metric by determining the proportion of queries wherein the associated code snippets are encompassed within the top-k entities of the resultant list.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{Rank}_i} \tag{11}$$

$$R@k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \delta(\text{Rank}_i \leq k) \tag{12}$$

In this context, Rank_i denotes the ranking of the code snippet that is paired with the i-th query within the resultant list. The function δ serves as an indicator, producing a value of 1 if $\text{Rank}_i \leq k$ and 0 otherwise.

State-of-the-art **CodeBERT (Feng et al. 2020b)**: Developed using the Transformer-based architecture and trained with a hybrid objective incorporating replaced token detection, CodeBERT efficiently leverages both bimodal data (NL-PL pairs) and unimodal data.

CodeRetriever (Li et al. 2022b): CodeRetriever incorporates two contrastive learning schemes: unimodal contrastive learning, which employs an unsupervised approach to build semantically-related code pairs based on documentation and function names, and bimodal contrastive learning, which utilizes documentation and inline comments to form code-text pairs.

CoCoSoDa (Shi et al. 2023b): CoCoSoDa is a novel approach for code search, leveraging multimodal momentum contrastive learning and soft data augmentation to retrieve semantically relevant code snippets from natural language queries.

Research Questions

- **RQ-1: How effective is *HyCoQA* in code search?**
- **RQ-2: What is the impact of key design choices on the performance of *HyCoQA*?**
- **RQ-3: How do visualizations of HyCoQA's representations differ between positive and negative pairs across programming languages?**
- **RQ-4: How does *HyCoQA* perform in real case?**

Experiment Results

[RQ-1]: Effectiveness of HyCoQA

[Experiment Goal (RQ-1)]: In this study, our primary objective is to rigorously assess and benchmark the performance of our newly proposed HyCoQA model, especially in the context of code search tasks. We have chosen the CodeSearchNet dataset for this evaluation due to its comprehensive coverage across six distinct programming languages. By pitting HyCoQA against widely recognized and state-of-the-art benchmarks such as CodeBERT, CoCoSoDa, and CodeRetriever, we intend to draw informed comparisons and insights about its relative strengths and potential areas for improvement. Given the intricate nature of code retrieval and its implications for developer productivity, we emphasize the Mean Reciprocal Rank (MRR) as our primary metric of evaluation. Through this, we aspire to understand and quantify the tangible improvements and benefits that HyCoQA might offer over existing models in the domain.

[Experiment Results (RQ-1)]:

In our rigorous assessment of model performance on the CodeSearchNet dataset, spanning six diverse programming languages, we juxtaposed the capabilities of our proposed model, HyCoQA, against three state-of-the-art benchmarks: CodeBERT, CoCoSoDa, and CodeRetriever. It's important to highlight that, in the absence of accessible results for CodeRetriever and considering the substantial resources required for its replication across the six programming languages, we undertook the task of reproducing CodeRetriever results for a comprehensive comparison.

The outcomes, as depicted in Table 1, are predominantly based on the MRR metric, chosen for its conciseness and the constraints imposed by space limitations. An inspection of the table reveals that HyCoQA exhibits a commendable performance, consistently outstripping the MRR scores of its counterparts across all programming languages. Specifically, when contrasted with CoCoSoDa, which is one of the most competitive benchmarks, HyCoQA demonstrates an improvement ranging from 3.5% to 4% across different languages, with an average enhancement of approximately 3.5%. Such consistent and tangible increments in MRR values underscore the efficacy and robustness of our HyCoQA model. Moreover, while CodeRetriever itself is a formidable contender, our HyCoQA model surpasses it by marginal yet consistent increments, solidifying its position as a leading model in this domain.

Table 1: Performance assessment of our methods is based on established metrics, with JS representing JavaScript. For the experiments, we maintained statistical significance at $p < 0.01$.

Model	Ruby	JS	Go	Python	Java	PHP	Avg.
CodeBERT	0.679	0.621	0.885	0.672	0.677	0.626	0.693
CoCoSoDa	0.818	0.764	0.921	0.757	0.763	0.703	0.788
CodeRetrieval	0.838	0.784	0.941	0.777	0.783	0.723	0.808
HyCoQA	0.853	0.799	0.956	0.792	0.798	0.738	0.823

We also conduct recall experiment across all baselines. The table provides a detailed comparison of various models' performance using the Recall metric across six programming languages. Recall measures a model's ability to identify relevant items, and a higher value indicates better retrieval of pertinent items. CodeBERT, a renowned benchmark, showcases consistent recall scores across all languages, with Go and Java being particularly impressive. However, CoCoSoDa enhances upon CodeBERT, especially in Ruby and JavaScript, as evidenced by its higher R@1 metric. Interestingly, CodeRetriever surpasses CoCoSoDa across all metrics and languages, emphasizing its superior capability in retrieving a more extensive set of relevant items. Yet, the standout performer is HyCoQA, which consistently outperforms all other models across every metric and programming language. This supe-

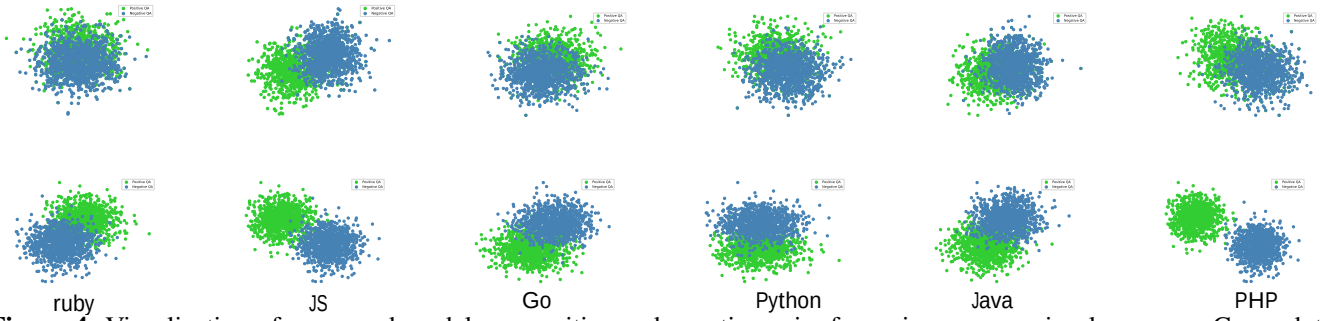

Figure 4: Visualization of compared models on positive and negative pairs from six programming languages. Green dots represent "positive QA" and blue dots means "negative QA". The first Row is visualized results of CodeRetriever and The second row is HyCoQA's.

rior performance positions HyCoQA as a potential leader in code search tasks, underlining its robustness and adaptability across various coding languages.

Table 2: Comparison on Recall Metric.

Models	Metric	Ruby	JavaScript	Go	Python	Java	PHP
CodeBERT	R@1	0.583	0.514	0.837	0.574	0.580	0.520
	R@5	0.800	0.752	0.944	0.792	0.796	0.753
	R@10	0.853	0.814	0.962	0.850	0.852	0.814
CoCoSoDa	R@1	0.655	0.582	0.861	0.614	0.624	0.561
	R@5	0.875	0.806	0.962	0.834	0.843	0.798
	R@10	0.916	0.866	0.978	0.888	0.890	0.863
CodeRetriever	R@1	0.665	0.592	0.871	0.624	0.634	0.571
	R@5	0.885	0.816	0.972	0.844	0.853	0.808
	R@10	0.926	0.876	0.988	0.898	0.900	0.873
HyCoQA	R@1	0.675	0.602	0.881	0.634	0.644	0.581
	R@5	0.895	0.826	0.982	0.854	0.863	0.818
	R@10	0.936	0.886	0.998	0.908	0.910	0.883

✍ **Answer to RQ-1:** *The experimental results accentuate the potential of HyCoQA in delivering superior performance in code search tasks across a gamut of programming languages. Experimental results indicate HyCoQA outperforms previous works and achieve a 3.5-4% improvement across languages in the term of MRR against the SOTA.*

[RQ-2]: Ablation Study

[**Experiment Goal**]: We perform an ablation study to investigate the effectiveness of each component in HyCoQA. The major novelty of ***HyCoQA*** is the fact that it explicitly includes and processes: ***hp*** hyperbolic representation. In addition, we also evaluate the component of BERT.

[**Experiment Design**]: We investigate the related contribution of *hp* and *bert* by building two variants of HyCoQA where we remove either *hp* (i.e., denoted as HyCoQA $_{hp-}$), or *bert* (i.e., denoted as HyCoQA $_{bert-}$). We evaluate the performance of these variants on the task of code search.

[Experiment Results (RQ-2)]:

The performance dynamics of ***HyCoQA*** are intricately tied to its constituent components. To shed light on the contribution of each individual component, we conducted an ablation study. The results, as illustrated in Table 3, pave the way for several enlightening insights. From the table, we observe that the absence of the hyperbolic representation component, denoted as **hp**, in ***HyCoQA***$_{hp-}$ results in a noticeable performance degradation. The average score drops to 0.787, representing a decline of approximately 4.38% in comparison to the comprehensive ***HyCoQA*** model. On the other hand, when we omit the BERT component, leading to ***HyCoQA***$_{bert-}$, the performance reduction is more modest. The average score settles at 0.818, a diminution of about 0.61%. This indicates that while BERT plays a contributory role, it's the hyperbolic representation that stands out as the linchpin in enhancing the model's efficacy. In summary, the complete ***HyCoQA*** model, which amalgamates both BERT and hyperbolic representation, attains the best performance metrics. This underscores its robustness and adaptability in tackling code retrieval tasks across a spectrum of programming languages. The ablation study provides a roadmap for future research, highlighting areas of potential improvement and innovation.

Table 3: Ablation Study

Model	Ruby	JS	Go	Python	Java	PHP	Avg.
HyCoQA$_{bert-}$	0.848	0.794	0.951	0.787	0.793	0.733	0.818
HyCoQA$_{hp-}$	0.830	0.760	0.920	0.752	0.758	0.700	0.787
HyCoQA	0.853	0.799	0.956	0.792	0.798	0.738	0.823

✍ **Answer to RQ-2:** *he ablation study of* **HyCoQA** *highlighted hyperbolic representation (**hp**)'s crucial role. Its absence resulted in a substantial 4.38% performance drop, while excluding the BERT component led to a mere 0.61% decline. The integrated* **HyCoQA** *model, which combines both elements, showcased superior performance.*

[RQ-3]: Visualization of Learned Representation

[Experiment Goal (RQ-3)]: In our study, we train our model to maximize the margin between scores for correct and negative QA pairs, using visualization to understand code retrieval capabilities. Efficient QA visualization becomes a metric to assess model performance. We compare SOTA CodeRetriever with our HyCoQA on six language-description QA pairs in this experiment.

[Experiment Results (RQ-3)]:

In our quest to comprehensively understand the discriminative capabilities of code retrieval models, we visualized the spatial distributions of positive and negative QA pairs. The underlying rationale behind this visualization is rooted in our training approach: our objective was to accentuate the margin between the scores of the correct QA pair and its negative counterpart. A clear demarcation between these two sets, when visualized, serves as a testament to the model's ability to efficiently retrieve relevant code snippets. A model's

prowess in code retrieval can, thus, be gauged by the clarity and distinction it offers in such visualizations.

For a comparative perspective, we elected two models for this visualization task: the state-of-the-art model CodeRetriever and our proposed model, HyCoQA. Our observations from the visualizations across all language-description QA pairs were illuminating. HyCoQA demonstrated an evident superiority in distinguishing between positive QA and negative QA pairs. The distinct clusters formed by HyCoQA were more segregated than those of CodeRetriever, underscoring its enhanced code retrieval capability. This clear visual distinction buttresses our assertion that HyCoQA possesses a heightened ability to discern and retrieve relevant code based on given queries, outshining its contemporaries in this domain.

✍ **Answer to RQ-3:** *Through visualizing positive and negative QA pairs, we evaluated code retrieval models' discriminative capabilities. While our training aimed to widen score margins between QA pairs, clarity in visualization proved the true test. Compared to the state-of-the-art CodeRetriever, HyCoQA excelled in discerning QA pairs across programming languages, highlighting its advanced code retrieval proficiency.*

[RQ-4]: Case Study

To further elucidate the superiority of our model ***HyCoQA***, we conducted a case study where we juxtaposed the predictions made by our model against those by the state-of-the-art baselines, namely CoCoSoDa, CodeBERT, and CodeRetriever.

Given the prompt: "a static method creating a Function from T to U using a given value", the ground truth code snippet is:

```
public static <T, U> Function<T, U>
    justFunction(U value) {
    return new JustValue <T, U> (
    value);
}
```

HyCoQA successfully predicts the ground truth while CodeRetriever's prediction:

```
public static <X> Processor
    setupFunction(X xInput) {
    return new DifferentClass(xInput);
}
```

CoCoSoDa's prediction:

```
public static <X, Y, Z> BiFunction<X, Z,
    Y> createComplexFunction(Y yParam, Z
    defaultParam) {
    ...
            return new
    AnotherFunctionClass<X, Y, Z>(yParam,
    zValue).apply(xValue, zValue);
        }
    };
}
```

In this instance, it is evident that ***HyCoQA*** offers a more accurate prediction in comparison to the baselines. Such cases underscore the robustness and precision of our model in understanding and generating code based on natural language descriptions.

✍ **Answer to RQ-4:** *In our case study,* **HyCoQA** *precisely retrieved a description with a correct code, outperforming baselines like CoCoSoDa and CodeRetriever in accuracy and recall.*

Related Work

Advancements in Code Representation

Code representation learning is pivotal for numerous software engineering tasks like code summarization (Iyer et al. 2016; LeClair, Jiang, and McMillan 2019; Shi et al. 2022), code search (Gu, Zhang, and Kim 2018a; Li et al. 2020; Haldar et al. 2020; Du et al. 2021), and more. Particularly, code search aids significantly in software development and maintenance (Singer et al. 2010; Nie et al. 2016). While early methods (McMillan et al. 2011b), (Lu et al. 2015; Lv et al. 2015b) leaned on lexical information retrieval, recent deep learning models embrace neural networks to enhance semantic code comprehension. Notable contributions include the use of sequential models (Wan et al. 2019), convolutional networks (Li et al. 2020), tree neural networks (Wan et al. 2019), graph models (Wan et al. 2019; Ling et al. 2021), and transformers (Du et al. 2021; Zhu et al. 2021). Large-scale pre-trained models (Guo et al. 2021; Feng et al. 2020a; Guo et al. 2022), (Niu et al. 2022) further enrich code semantics understanding, with exemplars like CodeBERT and GraphCodeBERT. Our method complements such pre-trained models, amplifying their efficacy.

Neural Interactions in QA and Hyperbolic Potential

While neural encoders like CNN or LSTM have proven their mettle in ranking models, recent focus gravitates towards the interaction layer. Initial models combined question and answer (QA) embeddings directly. Modern techniques, however, exploit similarity matrices, capturing nuanced word matches between QAs. Yet, these models, like AI-CNN or AP-BiLSTM (Xu et al. 2017; He, Gimpel, and Lin 2015; Shuai et al. 2020; Severyn and Moschitti 2015; Tay et al. 2017; Tay, Tuan, and Hui 2018a; Yu et al. 2014; Zhang et al. 2017; Tay, Tuan, and Hui 2018b), can be computationally demanding. Hyperbolic space offers an alternative, capturing hierarchical QA relationships efficiently.

Conclusion

In the evolving realm of software development, efficient code retrieval remains paramount. This study introduced the groundbreaking "Hyperbolic Code QA Matching" (HyCoQA), marking a significant stride in code search methodologies. By ingeniously harnessing the hierarchical representation capabilities of hyperbolic spaces and synergizing them with advanced embedding techniques, we've offered a solution that transcends traditional lexical matching. Our approach delves deeper, capturing the intrinsic semantic relationships between natural language descriptions and code. The empirical results underscore the superior efficacy of HyCoQA, setting a new benchmark in code search tasks. As the vast expanse of open-source platforms continues to

grow, tools like HyCoQA will become indispensable, empowering developers to navigate information oceans with unparalleled precision.

Acknowledgments

This work is supported by the NATURAL project, which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant No. 949014).

References

Bafna, P.; Pramod, D.; and Vaidya, A. 2016. Document clustering: TF-IDF approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 61–66. IEEE.

Bonnabel, S. 2013. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9): 2217–2229.

Cambronero, J.; Li, H.; Kim, S.; Sen, K.; and Chandra, S. 2019. When Deep Learning Met Code Search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, 964–974. New York, NY, USA: Association for Computing Machinery. ISBN 9781450355728.

Chai, Y.; Zhang, H.; Shen, B.; and Gu, X. 2022. Cross-Domain Deep Code Search with Meta Learning. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 487–498.

Cheng, Y.; and Kuang, L. 2022. CSRS: Code Search with Relevance Matching and Semantic Matching. arXiv:2203.07736.

Church, K. W. 2017. Word2Vec. *Natural Language Engineering*, 23(1): 155–162.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4171–4186.

Di Grazia, L.; and Pradel, M. 2023. Code Search: A Survey of Techniques for Finding Code. *ACM Comput. Surv.*, 55(11).

Du, L.; Shi, X.; Wang, Y.; Shi, E.; Han, S.; and Zhang, D. 2021. Is a single model enough? mucos: A multi-model ensemble learning approach for semantic code search. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2994–2998.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; and Zhou, M. 2020a. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547. Online: Association for Computational Linguistics.

Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. 2020b. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv preprint arXiv:2002.08155*.

Gu, X.; Zhang, H.; and Kim, S. 2018a. Deep Code Search. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, 933–944. New York, NY, USA: Association for Computing Machinery. ISBN 9781450356381.

Gu, X.; Zhang, H.; and Kim, S. 2018b. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering*, 933–944.

Guo, D.; Lu, S.; Duan, N.; Wang, Y.; Zhou, M.; and Yin, J. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7212–7225. Dublin, Ireland: Association for Computational Linguistics.

Guo, D.; Ren, S.; Lu, S.; Feng, Z.; Tang, D.; Liu, S.; Zhou, L.; Duan, N.; Svyatkovskiy, A.; Fu, S.; Tufano, M.; Deng, S. K.; Clement, C.; Drain, D.; Sundaresan, N.; Yin, J.; Jiang, D.; and Zhou, M. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. arXiv:2009.08366.

Haldar, R.; Wu, L.; Xiong, J.; and Hockenmaier, J. 2020. A multi-perspective architecture for semantic code search. *arXiv preprint arXiv:2005.06980*.

He, H.; Gimpel, K.; and Lin, J. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, 1576–1586.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 9729–9738.

Iyer, S.; Konstas, I.; Cheung, A.; and Zettlemoyer, L. 2016. Summarizing source code using a neural attention model. In *54th Annual Meeting of the Association for Computational Linguistics 2016*, 2073–2083. Association for Computational Linguistics.

Joulin, A.; Grave, E.; Bojanowski, P.; Douze, M.; Jégou, H.; and Mikolov, T. 2016. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

LeClair, A.; Jiang, S.; and McMillan, C. 2019. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the 41st International Conference on Software Engineering*, 795–806.

Li, W.; Qin, H.; Yan, S.; Shen, B.; and Chen, Y. 2020. Learning code-query interaction for enhancing code searches. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 115–126. IEEE.

Li, X.; Gong, Y.; Shen, Y.; Qiu, X.; Zhang, H.; Yao, B.; Qi, W.; Jiang, D.; Chen, W.; and Duan, N. 2022a. CodeRetriever: A Large Scale Contrastive Pre-Training Method for Code Search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2898–2910. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics.

Li, X.; Gong, Y.; Shen, Y.; Qiu, X.; Zhang, H.; Yao, B.; Qi, W.; Jiang, D.; Chen, W.; and Duan, N. 2022b. CodeRetriever: A Large Scale Contrastive Pre-Training Method for Code Search. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2898–2910.

Ling, X.; Wu, L.; Wang, S.; Pan, G.; Ma, T.; Xu, F.; Liu, A. X.; Wu, C.; and Ji, S. 2021. Deep graph matching and searching for semantic code retrieval. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(5): 1–21.

Linstead, E.; Bajracharya, S.; Ngo, T.; Rigor, P.; Lopes, C.; and Baldi, P. 2009. Sourcerer: Mining and Searching Internet-Scale Software Repositories. *Data Min. Knowl. Discov.*, 18(2): 300–336.

Liu, C.; Xia, X.; Lo, D.; Gao, C.; Yang, X.; and Grundy, J. 2021. Opportunities and Challenges in Code Search Tools. *ACM Comput. Surv.*, 54(9).

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

Lv, F.; Zhang, H.; Lou, J.-g.; Wang, S.; Zhang, D.; and Zhao, J. 2015a. 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)*, 260–270.

Lv, F.; Zhang, H.; Lou, J.-g.; Wang, S.; Zhang, D.; and Zhao, J. 2015b. 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)*, 260–270. IEEE.

McMillan, C.; Grechanik, M.; Poshyvanyk, D.; Xie, Q.; and Fu, C. 2011a. Portfolio: finding relevant functions and their usage. In *2011 33rd International Conference on Software Engineering (ICSE)*, 111–120.

McMillan, C.; Grechanik, M.; Poshyvanyk, D.; Xie, Q.; and Fu, C. 2011b. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering*, 111–120.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nickel, M.; and Kiela, D. 2017. Poincaré embeddings for learning hierarchical representations. *Advances in neural information processing systems*, 30.

Nie, L.; Jiang, H.; Ren, Z.; Sun, Z.; and Li, X. 2016. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing*, 9(5): 771–783.

Niu, C.; Li, C.; Ng, V.; Ge, J.; Huang, L.; and Luo, B. 2022. Spt-code: Sequence-to-sequence pre-training for learning source code representations. In *Proceedings of the 44th International Conference on Software Engineering*, 2006–2018.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Severyn, A.; and Moschitti, A. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, 373–382.

Shi, E.; Wang, Y.; Du, L.; Chen, J.; Han, S.; Zhang, H.; Zhang, D.; and Sun, H. 2022. On the evaluation of neural code summarization. In *Proceedings of the 44th International Conference on Software Engineering*, 1597–1608.

Shi, E.; Wang, Y.; Gu, W.; Du, L.; Zhang, H.; Han, S.; Zhang, D.; and Sun, H. 2023a. CoCoSoDa: Effective Contrastive Learning for Code Search. arXiv:2204.03293.

Shi, E.; Wang, Y.; Gu, W.; Du, L.; Zhang, H.; Han, S.; Zhang, D.; and Sun, H. 2023b. CoCoSoDa: Effective Contrastive Learning for Code Search. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2198–2210. IEEE.

Shuai, J.; Xu, L.; Liu, C.; Yan, M.; Xia, X.; and Lei, Y. 2020. Improving Code Search with Co-Attentive Representation Learning. In *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, 196–207. New York, NY, USA: Association for Computing Machinery. ISBN 9781450379588.

Singer, J.; Lethbridge, T.; Vinson, N.; and Anquetil, N. 2010. An examination of software engineering work practices. In *CASCON First Decade High Impact Papers*, 174–188.

Sun, W.; Fang, C.; Chen, Y.; Tao, G.; Han, T.; and Zhang, Q. 2022. Code Search Based on Context-Aware Code Translation. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, 388–400. New York, NY, USA: Association for Computing Machinery. ISBN 9781450392211.

Tang, X.; Tian, H.; Kong, P.; Liu, K.; Klein, J.; and Bissyande, T. F. 2023. App Review Driven Collaborative Bug Finding. *arXiv preprint arXiv:2301.02818*.

Tay, Y.; Phan, M. C.; Tuan, L. A.; and Hui, S. C. 2017. Learning to rank question answer pairs with holographic dual lstm architecture. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, 695–704.

Tay, Y.; Tuan, L. A.; and Hui, S. C. 2018a. Cross temporal recurrent networks for ranking question answer pairs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Tay, Y.; Tuan, L. A.; and Hui, S. C. 2018b. Hyperbolic representation learning for fast and efficient neural question answering. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 583–591.

Wan, Y.; Shu, J.; Sui, Y.; Xu, G.; Zhao, Z.; Wu, J.; and Yu, P. 2019. Multi-modal attention network learning for semantic source code retrieval. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 13–25. IEEE.

Xu, L.; Yang, H.; Liu, C.; Shuai, J.; Yan, M.; Lei, Y.; and Xu, Z. 2021. Two-Stage Attention-Based Model for Code Search with Textual and Structural Features. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 342–353.

Xu, S.; Cheng, Y.; Gu, K.; Yang, Y.; Chang, S.; and Zhou, P. 2017. Jointly attentive spatial-temporal pooling networks for video-based person re-identification. In *Proceedings of*

the IEEE international conference on computer vision, 4733–4742.

Yu, L.; Hermann, K. M.; Blunsom, P.; and Pulman, S. 2014. Deep learning for answer sentence selection. *arXiv preprint arXiv:1412.1632*.

Zhang, X.; Li, S.; Sha, L.; and Wang, H. 2017. Attentive interactive neural networks for answer selection in community question answering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Zhu, Q.; Sun, Z.; Liang, X.; Xiong, Y.; and Zhang, L. 2021. OCoR: An Overlapping-Aware Code Retriever. ASE '20, 883–894. New York, NY, USA: Association for Computing Machinery. ISBN 9781450367684.