Code Compliance Assessment as a Learning Problem

Neela Sawant
AWS AI, Amazon
Bengaluru, India
nsawant@amazon.com

Srinivasan H. Sengamedu
AWS AI, Amazon
Seattle, USA
sengamed@amazon.com

***Abstract*—Manual code reviews and static code analyzers are the traditional mechanisms to verify if source code complies with coding policies. However, they are hard to scale. We formulate *code compliance assessment* as a machine learning (ML) problem, to take as input a natural language policy and code, and generate a prediction on the code's compliance, non-compliance, or irrelevance. Our intention for ML-based automation is to scale the development of *Amazon CodeGuru*, a commercial code analyzer. We explore key research questions on model formulation, training data, and evaluation setup. We obtain a joint code-text representation space (embeddings) which preserves compliance relationships via the vector distance of code and policy embeddings. As there is no task-specific data, we re-interpret and filter commonly available software datasets with additional pre-training and pre-finetuning tasks that reduce the semantic gap. We benchmarked our approach on two listings of coding policies (*CWE* and *CBP*). This is a zero-shot evaluation as none of the policies occur in the training set. On *CWE* and *CBP* respectively, our tool *Policy2Code* achieves classification accuracies of (59%, 71%) and search MRR of (0.05, 0.21) compared to CodeBERT with classification accuracies of (37%, 54%) and MRR of (0.02, 0.02). In a user study, 24% Policy2Code detections were accepted compared to 7% for CodeBERT. *Policy2Code* is considered a useful ML-based aid to supplement manual efforts.**

***Index Terms*—code embeddings, natural language analysis**

I. Introduction

Coding policies are natural language rules or best practices on diverse topics like security, exception handling, and input validation. Code compliance assessment, i.e. verifying that source code is compliant with coding policies, is an important but non-trivial problem. Consider two policies from the well-known *Common Weakness Enumeration* (CWE) listing [1].

1) CWE-396[1] - *"Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities"*. Now consider this code snippet:

```
private static byte[] decrypt(byte[] src, byte[] key)
    throws RuntimeException {
    try {
        SecureRandom sr = new SecureRandom();
        DESKeySpec dks = new DESKeySpec(key);
        ...
        Cipher cipher = Cipher.getInstance(DES);
        cipher.init(Cipher.DECRYPT_MODE, securekey, sr)
            ;
        return cipher.doFinal(src);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

This code is non-compliant with CWE-396 as the *'catch (Exception e)'* statement broadly covers all exception types. Instead, specific exceptions such as *NoSuchAlgorithmException* or *NullPointerException* should be used.

2) CWE-197[2] - *"Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion"*. Now consider this code snippet:

```
public static byte[] toByteArray(InputStream input,
    long size) throws IOException {
    if (size > Integer.MAX_VALUE) {
        throw new IllegalArgumentException(""Size
            greater than Integer max value: "" + size)
            ;
    }
    return toByteArray(input, (int) size);
}
```

This code is compliant with CWE-197 because the casting operation *'(int) size'* maps variable *size* from primitive *long* to another primitive *int* of smaller size, and this operation is protected against truncation errors via the preceding *if* comparison on integer maximum value.

Code compliance assessment requires a deep understanding of program behavior and associated policies. This task is subsequently carried out via manual code reviews and static code analysis. However, both approaches are hard to scale to increasing and evolving coding policies across diverse programming languages and frameworks. The global market size for static code analyzers is projected to reach US $2002 million by 2027, from US $748.1 million in 2020, at a cumulative annual growth rate of 15.1% during 2021-2027 [2]. Increased automation can help with this goal.

Our objective is to explore how machine learning can help scale code compliance assessment - *"Can we automatically label a code as compliant or non-compliant (or irrelevant) given any natural language policy input?"* Our primary use-case is *Amazon CodeGuru* (https://aws.amazon.com/codeguru/) [3], a developer tool that provides intelligent recommendations to improve code quality and identify an application's most expensive lines of code. Automation in code compliance assessment can scale the development of *Amazon CodeGuru* in two ways: (a) *classification* - detecting non-compliant codes for policies not covered by manual reviews and existing static analysis rules, and (b) *search* - finding compliant and non-compliant code examples that will serve as test cases in developing new static analysis rules. While there is prior work

[1]https://cwe.mitre.org/data/definitions/396.html

[2]https://cwe.mitre.org/data/definitions/197.html

in detecting buggy codes [4], [5], [6], [7] as well as natural language code search [8], [9], [10], no solutions exist today that can simultaneously determine code relevance as well as compliance and non-compliance. In this paper, we set up a learning framework and explore three research questions.

- *Learning objective* - Compliance assessment is similar, but more fine-grained than relevance assessment. Both the compliant and non-compliant examples are relevant to a coding policy. *How can we frame the learning problem?* We propose learning different embeddings to represent compliant and non-compliant *facets* of natural language policies. We formulate a representation learning approach where the relationships between the compliant and non-compliant policy facets and compliant, non-compliant, and irrelevant code examples are preserved via the vector distances between their embeddings. Representation learning can scale compliance assessment in a *zero-shot* setting [11] as new policies and code examples can be mapped to their embeddings, even if they are not part of the original training dataset.
- *Training setup* - There are no task-specific labeled training datasets on coding policies associated with their examples. Curating new training datasets can be prohibitively expensive. *What training data can be used?* We propose repurposing general-purpose code and documentation datasets and propose additional training and filtering tasks to reduce the semantic gap.
- *Evaluation setup - How to quantify the performance of learning-based tools for code compliance assessment?* We create a benchmark that reinterprets and repurposes well known listings of coding issues curated by program analysis experts: (a) Common Weaknesses Enumeration (CWE) [1] and (b) Coding Best Practices (CBP) from Amazon CodeGuru test suite [3]. We consider weakness or issue descriptions as policies, buggy examples as non-compliant and correct code examples as compliant examples. For realistic search setting, we further add 27K unlabeled code snippets from the CodeSearchNet test dataset [12]. We use standard metrics to measure performance (*accuracy* for classification and *mean reciprocal rank (MRR)* for search) to compare against two strong baselines - CodeBERT and a multi-class model.

We develop a new tool *Policy2Code* by exploring policy representations, loss formulation, and training schemes. On CWE and CBP respectively, *Policy2Code* achieves classification accuracies of (59%, 71%) and search MRR of (0.05, 0.21). In comparison, CodeBERT achieves classification accuracies of (37%, 54%) and MRR of (0.02, 0.02) respectively. The multi-class model achieves classification accuracies of (54%, 60%) and MRR of (0.03, 0.13) respectively. We set up Policy2Code search engine to on-demand retrieve compliant and non-compliant examples of new policies. A user study with program analysis experts showed that 24% detections from *Policy2Code* were accepted compared to 7% detections from CodeBERT and 15% by the multi-class model. *Policy2Code* is better than other ML alternatives, but still not practical as a stand alone code compliance tool. We expect it to augment manual efforts in the near future. We hope our work spawns future research in alternative and robust formulations for automated code compliance assessment.

II. Related Work

We discuss machine learning for code applications as well as key ideas in information retrieval and representation learning.

A. Machine Learning for Code

Machine learning and deep learning in particular, have advanced many software engineering tasks such as variable naming [13], comment understanding [14], bug detection [4], code review generation [15], and code and documentation synthesis [16], [17]. Deep learning research has benefitted from efficient transformer models [18] such as CodeBERT [19] - a 124M parameter text-code encoder with impressive performance on documentation generation and code search tasks. Another useful concept is that of pre-training, allowing data-hungry deep learning models to learn from large unlabeled or weakly labeled datasets on related tasks, so that the models can be subsequently fine-tuned using small task-specific training datasets [20], [21], [22]. CodeBERT is trained with CodeSearchNet training dataset for two tasks: masked language modeling and code-text relevance (method body to method descriptions).

Neural Bug Finding is typically modeled in a supervised setting using labeled training examples for a pre-determined set of issues. Training examples may be obtained from existing static analyzer detections [4] or synthesized with transformations [5]. Much of the research is devoted to analyzing the effect of richer program representations. For example, Wang et al. [6] show that combining different types of contexts (the Program Dependence Graph (PDG), Data Flow Graph (DFG) and the method under investigation) improves performance of bug detection in pre-determined categories.

Earlier *Code Search* systems predominantly relied on string matching, and in some cases aiming to improve performance with code structures [23], API matching [24], API usage patterns [25], [26], and query expansion [27]. More recent neural code search systems perform search based on the vector distance between their embeddings. This research focuses on three aspects (a) code representations, for example using method names and API sequences [10], parse trees [28], and control flow graphs [29], [30], (b) training strategy, for example, unsupervised [8], [31] or supervised [28], [19], [32], [33], and (c) loss formulations, for example, contrastive loss or [28], [19] or triplet loss [10], [29], [30], [33], [34].

B. Representation learning

To the best of our knowledge, there are no settings for fine-grained compliance assessment of code and policy pair. We review key research in general information retrieval and representation (metric) learning [35], [36], [37], [38]. Representation learning has the attractive property that once

the embedding space is learned, novel input can be mapped without additional training, essentially a *zero-shot* setting [11].

A common ranking mechanism is *triplet loss* that models relationships among three data points - an anchor, a positive example, and a negative example. In traditional code search, they map to natural language query (such as a method description) to relevant example (such as the corresponding method body) and an irrelevant example (such as the body of an unrelated method). [39], [40], [41]. The algorithm learns to minimize the distance between anchor-positive pairs and maximize the distance between the anchor-negative pairs. Triplet mining is an active area of research [39], [40], [41]. There are very few settings involving higher order losses for fine-grained differentiation. For example, Chen et al. [42] introduced a quadruplet loss network for person verification task. A training example in this task consists of four data points x_i, x_j, x_k, x_l where x_i, x_j belong to the same class (e.g., considered the same person) and x_k and x_l belong to two other classes (faces from two other people).

III. Learning Problem Formulation

Let r and c denote a natural language policy and a code example. Let $y \in \{+, -\}$ denote the compliant and non-compliant facets respectively, and let irrelevance be denoted by $\sim$. Our objective is to learn a bimodal embedding space for r and c such that the the compliant, non-compliant (and irrelevance) relationships are represented via the vector distances between their embeddings. In contrast to code search, where natural language query has a single representation, we propose to learn separate r^+ and r^- embeddings for fine-grained differentiation. For compliant and non-compliant example search, we then operate with two separate query inputs r^+ and r^-. Embedding of r^+ should be closer to c^+, followed by c^-, and dissimilar to that of $c^\sim$. For non-compliant search, r^- is closer to c^-, followed by c^+, and far from $c^\sim$. Such space naturally supports search and classification based on relative distances and learned thresholds. New policies and codes can be mapped to their embeddings and evaluated in zero-shot setting. We now discuss two aspects of representation learning - policy input representation and loss functions.

A. Facet Policy Input Representation

Policy representation is a complex problem. A policy contains a general relevance context and a specific compliance context. For example, reconsider *(CWE-396): Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities*. This policy is relevant to all code snippets with exception handling. The compliance context refers to using broad exception types (*Exception*) versus more specific exceptions (*NoSuchAlgorithmException* or *NullPointerException*). Policy representations can incorporate domain knowledge and advanced parsing of natural language text. In the context of this paper however, we explore two purely learning based variations inspired from prior work in NLP and Computer Vision.

1) Facet-prefixed Policy: This idea is similar to T5 query format [43] with remarkable success in NLP applications. Natural language policy description r is prefixed with facet token y. The concatenated representation $y : r$ is passed through an encoder to generate faceted policy embedding. We use a transformer-based encoder with self-attention, so that the facet token organically attends to the various policy text components.

2) Facet-masked Policy: This approach externally combines the facet token with the policy input using a parametric model. We use an implementation of conditional masking [44], which aims to learn as many mask vectors as the number of conditions (e.g., facets in our setting). One of the mask vector is selected based on the facet y and multiplied with embedding of policy r, essentially projecting the policy into a different subspace conditioned on the facet label. See Appendix A for details.

B. Loss Functions

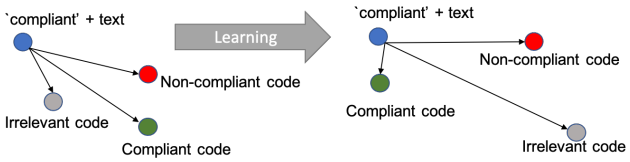

Fig. 1: Illustration of compliance search task (with compliant facet): The learning objective is to minimize the vector distance between the faceted policy description and a compliant code embedding, and maximize the distance with respect to non-compliant and irrelevant code embeddings.

Figure 1 illustrates the learning objective using an example of compliance search task. Given a training quadruplet $(r^+, c^+, c^-, c^\sim)$, we want to learn embeddings that minimize the relative distances between matched pairs and maximize the distances between unmatched pairs. We test two formulations.

1) Quadruplet Loss (QL) : Quadruplet loss is an extension of triplet loss computed as follows:

$$L^+_{quad} = \sum_{r^+,c^+,c^-}^{N} [d(f_{r^+}, f_{c^+})^2 - d(f_{r^+}, f_{c^-})^2 + \alpha_1]_+ + \sum_{r^+,c^-,c^\sim}^{N} [d(f_{r^+}, f_{c^-})^2 - d(f_{r^+}, f_{c^\sim})^2 + \alpha_2]_+ \quad (1)$$

where f_x denotes the embedding of sequence x, $d(.,.)$ denotes the distance function between two embeddings, and $[]_+$ indicates that only positive values contribute towards the loss calculation. $\alpha_2 > \alpha_1 > 0$ are tunable margin parameters for compliance and relevance differentiation, respectively. The loss for the non-compliant facet can be computed similarly over quadruplet $(r^-, c^-, c^+, c^\sim)$. Finally, the total loss is computed as $L_{quad} = \frac{1}{2}(L^+_{quad} + L^-_{quad})$.

The naive quadruplet loss formulation has two limitations. Eqn. 1 is anchored around text only. It does not encode constraint for classification task because there is no anchoring around code and no constraint to ensure that embeddings of r^+ and r^- are different given a code. Another limitation is from

practically available data. In general purpose datasets, very few examples will contain both compliant and non-compliant codes. However, there is an abundance of datasets with partial view (some contain only relevant examples, some with only compliant or non-compliant examples, etc.). To be able to use the latter datasets, we need an alternative formulation as below.

2) Bimodal Multi-task Triplet loss (BMT) : In this formulation, the quadruplets $(r^+, c^+, c^-, c^\sim)$ and $(r^-, c^-, c^+, c^\sim)$ are un-pivoted into a batch B of labeled examples (x, l), where x can be either a text or code (e.g., x is bimodal) and label l is generated on the fly such that only the x originating from paired policy and facets share that label l. For example, the quadruplets mentioned above can be un-pivoted into a list $[(r^+, l^1), (c^+, l^1), (r^-, l^2), (c^-, l^2), (c^\sim, l^\sim), \ldots]$. Each irrelevant example $c^\sim$ is assigned a unique label $l^\sim$ to prevent anchor-positive matching with another entity. We then calculate a *bimodal multi-task triplet (BMT)* loss over possible valid triplets (a, p, n) where a, p, and n are anchor, positive, and negative sequences in the batch such that a and p share the same label and n has a different label. The term *multi-task* denotes the simultaneous modeling of different class labels.

$$L_B = \sum_{(a,p,n);l(a)=l(p),l(a)\neq l(n)} [d(f_a, f_p)^2 - d(f_a, f_n)^2 + \alpha]_+ \tag{2}$$

where α is a common margin which acts as a threshold to separate the irrelevant examples from the relevant ones. The formulation naturally allows both code and text entities to serve as anchors. Datasets with partial labels (only relevant, compliant, and/or non-compliant) can be incorporated into training. Compliance search task is served by selecting codes based on their distances from faceted policy representations. For compliance classification, we use the threshold α to determine if a code c is relevant or irrelevant with respect to the average policy embedding $(r^+ + r^-)/2$. If it is deemed relevant, we use the distances with respect to r^+ and r^- to determine the closest facet as compliance label. The distances can be mapped to probabilities using SoftMax function $p(d) = \frac{e^{-d}}{\sum_{d'} e^{-d'}}$.

IV. Training Setup

ML models are data hungry. For code compliance assessment however, there are no large-scale datasets containing natural language policies paired with their compliant and non-compliant examples. We re-interpret and repurpose general datasets on code, documentation, code reviews, and bug-fixes with additional training schemes to bridge this gap.

A. Multi-Task Pre-training

To capture patterns in higher level documentation guidelines, pre-training with general documentation can be helpful. To capture coding issues and aligned code examples, pre-training with code reviews can be beneficial. We use below tasks.

- *Collocated documentation prediction* (Doc): Paragraphs of software documentation are segmented into non-overlapping fixed-length passages. The passages belonging to the same paragraph are assigned a common label. The objective of *Collocated documentation prediction* is to predict which candidate segments (code snippet or text) in a batch come from the same paragraph.
- *Code-comment matching task* (CC): Given multiple <code, review comment> pairs, we attempt to determine (a) the correct review comment given a code (*relevant comment prediction*) and (b) the correct code on which a review comment is made (*non-compliant code prediction*).

B. Pre-fine-tuning

Examples containing bug-fixes are especially valuable to us, because the example can be reinterpreted as a proxy to policy description with compliant and non-compliant codes. Such a dataset can then be used to pre-finetune for target task. See Figure 2 for a motivating example of a GitHub bug-fix review comment. The comment provides a security best practice suggestion for which the code-before and code-after versions can be used as non-compliant and compliant examples, respectively. Each review comment is essentially treated as a policy and the code-before and the code-after versions are taken to be the non-compliant and the compliant code examples respectively. Irrelevant examples $c^\sim$ are mined from codes of unrelated bug-fixes. This dataset can be noisy, but we expect useful signals to emerge with large-scale datasets.

Data Filtering: As an additional treatment, we apply *Doc2BP* [45], a deep learning classifier trained to detect coding policies and best practice recommendations from natural language text. *Doc2BP* analyzes keywords, parts of speech patterns, and other linguistic properties to determine if a comment resembles coding policies, and filters out non-policy-like comments to reduce noise. Results with and without *Doc2BP* indicate that the filtering step in pre-finetuning is useful.

V. Evaluation Setup

We construct a new benchmark by repurposing two listings of coding issues and a public code search dataset.

- Common Weaknesses Enumeration (*CWE*) View 702 [1] is a public domain list of software weaknesses introduced during implementation. It contains 185 Java weaknesses, each with a short natural language description and compliant and non-compliant examples. We use the weakness description as the policy. The number of policies with non-compliant, compliant, and both types of examples are 182, 36, and 33, respectively.
- Coding Best Practice (*CBP*) is a collection of Java best practice descriptions and compliant and non-compliant examples obtained from Amazon CodeGuru [3] static analyzer test-suite. Our dataset contains 46 Java best practices, each with at least one compliant and one non-compliant example, and a total of 186 examples.
- CodeSearchNet (*CSN*) is a public domain dataset [12]. We use the test split with 27K unlabeled code examples.

The resulting benchmark contains *231 policies*, *477 labeled codes*, and *27K unlabeled codes* (total 28K code examples). Policies cover many topics like input validation, cipher security, web security, command injection, preferred data structures, etc.

Code review comment: *Probably better to use 'SecureRandom' for anything security related.*

Facet: non-compliant/ Code-before

```
private String generatePassword(){
  String allowedChars="
       abcdefghijklmnopqrstuvwxyz0123456789";
  int charRange=allowedChars.length();
  Random random=new Random();
  StringBuilder password=new StringBuilder();
  int passwordLength=10;
  for (int i=0; i < passwordLength; i++) {
    password.append(allowedChars.charAt(random.nextInt(
        charRange)));
  }
  return password.toString();
}
```

Facet: compliant/ Code-after

```
private String generatePassword() throws
     NoSuchAlgorithmException {
  String allowedChars="
       abcdefghijklmnopqrstuvwxyz0123456789";
  int charRange=allowedChars.length();
  SecureRandom random=SecureRandom.getInstanceStrong();
  int passwordLength=20;
  char[] password=new char[passwordLength];
  for (int i=0; i < passwordLength; i++) {
    password[i]=allowedChars.charAt(random.nextInt(
        charRange));
  }
  return new String(password);
}
```

Fig. 2: Example entry in GitHub comment and bug-fix dataset

The dataset has no overlap with the training data, hence this is a zero-shot benchmark. We investigate two tasks:

- Compliance Classification: The objective is to predict whether a code is compliant, non-compliant, or irrelevant given a policy. Performance is measured in accuracy with respect to the known ground truth.
- Compliance Search: The objective is to retrieve compliant and non-compliant codes from a large code corpus for a user-specified natural language policy. In a large corpus setting, we do not expect to know the ground truth labels for all code examples. Hence we rely on another metric in information retrieval research, the mean reciprocal rank metric (MRR) defined as $\frac{1}{Q}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$ where Q denotes the number of user queries and $rank_i$ denotes the rank position of the first example from the known ground truth within each facet. We prefer MRR over other information retrieval metrics such as precision@k for its ability to capture the order of relevance. For example, a precision@5 metric would yield the same score (0.2) whether a single relevant result appears at rank 1 or rank 5. MRR on the other hand, would generate a score of 1 for the first position and a score of 0.2 for the fifth position.

We propose two competitive baselines.

- CodeBERT [19] is the off-the-shelf pre-trained HuggingFace implementation. CodeBERT is a state-of-the-art deep learning approach for several code-related tasks including code search (https://microsoft.github.io/CodeXGLUE/). Despite using a token sequence representation of code, it is able to out-perform complex models such as GraphCodeBERT that can represent programs as graphs. This baseline is designed for relevance and not fine-grained differentiation. It represents the current best tool available to developers to find relevant examples, which can then be manually classified for compliance.
- Multi-class classification is a three-class (compliant, non-compliant, irrelevant) classifier. It is implemented by adding dropout and linear layers on top of the CodeBERT's [19] transformer embedding layer. This is also trained for a zero-shot setting. A (policy, code) pair is represented using CodeBERT's input sequence convention $[CLS]x_1, x_2, ...x_m[SEP]y_1, y_2, ...y_n[SEP]$ where x and y represent the tokens for policy and code respectively.

Our use of public domain datasets and easy to replicate baselines is expected to fuel future research on this topic.

VI. Results

This section presents a comprehensive evaluation. **Bold**-facing in tables is used to indicate superior performance.

A. Experimental Setup

As the central bimodal code-text encoder for Policy2Code as well as the multi-class classification baseline, we use the HuggingFace implementation of CodeBERT [19]. The multi-class classification baseline and the Policy2Code model are trained using the same pre-training and pre-fine-tuning data for apples-to-apples comparison. For the *collocated documentation pre-training* (DOC) task, we use Java 8, 11, and 16 documentation ($\sim$1 GB). For *code comment matching pre-training* (CC), we used 150K GitHub code reviews. For *bug-fix-comment pre-fine-tuning*, we leveraged 32K code reviews on GitHub that were tagged as bug-fix, split into 80%-20% for training and validation. All these datasets are in public domain. Further none of them intentionally overlap with benchmark examples. Code is statically represented as token sequences with sub-word tokenization and BytePair Encoding (BPE) [46].

B. Policy2Code Model Training

We now discuss the effect of policy representations, losses, and training schemes. Models were optimized using automatic hyper-parameter tuning. See Appendix B for details.

Table I reports the performance of different combinations of loss functions and policy representations on the CWE and CBP datasets for classification and search tasks. *The BMT loss with facet-prefixed policy representation is the most effective strategy across both datasets and tasks.*

Table II shows the effect of different pre-training and filtering schemes on Policy2Code performance. All schemes use BMT loss with facet-prefixed policy representation and subsequently

TABLE I: Effect of Loss Formulation and Policy Representation

Loss	Policy Rep	Classification Accuracy			Search MRR		
		CWE (Conceptual)	CWE	CBP	CWE (Conceptual)	CWE	CBP
QL	Prefixed	42.6 ± 3.6	48.13 ± 2.2	44.93 ± 0.98	0.021 ± 0.00	0.0321 ± 0.009	0.0915 ± 0.017
	Masked	33.41 ± 1.9	34.97 ± 2.2	39.3 ± 6.3	0.0298 ± 0.02	0.0318 ± 0.19	0.07641 ± 0.029
BMT	Prefixed	**45.3** ± 2.4	**49.8** ± 5.1	**48.0** ± 7.1	0.0264 ± 0.029	**0.0363** ± 0.011	**0.1496** ± 0.021
	Masked	41.6 ± 4.5	42.23 ± 3.7	46.21 ± 1.2	0.0242 ± 0.061	0.0213 ± 0.032	0.1127 ± 0.027

TABLE II: Effect of Pre-training and Filtering Schemes

Pre-Training Scheme	Classification Accuracy			Search MRR		
	CWE (Conceptual)	CWE	CBP	CWE (Conceptual)	CWE	CBP
None	45.3 ± 2.4	49.8 ± 5.1	48.0 ± 7.1	0.0264 ± 0.029	0.0363 ± 0.011	0.1496 ± 0.021
Doc	48.3 ± 3.1	54.2 ± 4.6	56.2 ± 2.6	0.0319 ± 0.038	0.0313 ± 0.008	0.1773 ± 0.009
CC	47.8 ± 6.1	43.77 ± 1.1	55.2 ± 2.9	0.0471 ± 0.129	0.0467 ± 0.008	0.1654 ± 0.017
Doc + CC	53.2 ± 3.2	**59.3** ± 7.4	67.6 ± 7.9	**0.0492** ± 0.05	0.0482 ± 0.011	0.1982 ± 0.012
Doc + CC + *Doc2BP* Filter [45]	**53.8** ± 6.1	**59.3** ± 2.0	**71.0** ± 2.8	0.0462 ± 0.013	**0.0538** ± 0.01	**0.2182** ± 0.012

followed by the same pre-finetuning step. The filtering step reduces the dataset to 14K examples (out of 32K original bug-fix dataset) which are predicted to be policy-like. *The pre-training schemes as well as the filtering step improve model performance across tasks*. We continue to see better performance for the CBP dataset than the CWE dataset. We also observe that documentation and code-related pre-training impact CWE performance differently. While code-related pre-training (CC) decreases the CWE performance, documentation pre-training improves the performance. This effect can be explained as CWE policies are more conceptual, hence more similar to documentation patterns than specific code comments.

C. Baseline Comparison

Table III compares Policy2Code with baselines on compliance classification and compliance search tasks. Policy2Code outperforms CodeBERT as well as the multi-class classification baseline. Policy2Code achieves classification accuracies of (59%, 71%) and search MRR of (0.05, 0.21) on CWE and CBP. On the same datasets, a popular CodeBERT baseline achieves classification accuracies of (37%, 54%) and MRR of (0.02, 0.02) respectively. Multi-class classification achieves accuracies of (54%, 60%) and MRR of (0.03, 0.13) respectively.

We observe that the model performance is generally lower on the CWE dataset. Our investigation shows that CWE contains multiple policies on related topics and partially labeled ground truth. One code example may be applicable to multiple policies, but not all such relationships are recorded. For example, CWE-338[3] suggests *using cryptographically strong Pseudo-Random Number Generator (PRNG) in a security context*. The below code is the only explicitly labeled example for this policy.

```
Random random = new Random(System.currentTimeMillis());
int accountID = random.nextInt();
```

However, other code examples in the CWE dataset may also be related to CWE-338. See the below code for example, labeled only for policy CWE-336[4] on *not using the same seed across multiple PRNG initializations*. This example can also be considered as a valid example for CWE-338.

```
private static final long SEED = 1234567890;
public int generateAccountID() {
    Random random = new Random(SEED);
    return random.nextInt();
}
```

This implies that the CWE ground truth is incomplete, leading to currently computed metrics being a lower bound on the actual performance. CBP is expected to have higher quality as it is manually curated for compliance analysis task specifically.

D. User Study

We conducted a user study to determine if Policy2Code can be used in a practical problem of test case identification in developing static analysis rules. See Figures 3 and 4 for anecdotal examples. On the one hand, they may seem to over-index on keyword similarity (Figure 3(c), (e)), and on the other hand, we find surprisingly semantic associations. For example, Figure 3(a)) is the example discussed in the introduction where a compliant code is found for CWE-197.

We built a FAISS-GPU index [47] over all 28K code examples in the benchmark dataset. On a p3.xlarge EC2 machine, index creation takes approximately 8 seconds. At inference time, top examples for any policy queries are retrieved within milliseconds based on approximate and efficient nearest neighbor algorithm. Exact similarity computation is avoided as it can take about 30 minutes per policy to compare all codes.

We recruited 16 experts with an average of over 10 years of software engineering experience and over 5 years of programming language analysis experience. For 25 policies in CWE dataset, we determined the top 5 compliant and non-compliant examples detected by various approaches in the entire 28K code example benchmark dataset. We masked the identity of the selection algorithm and asked the assessor if they agree or disagree with the label assignment. Table IV shows the acceptance rate of predictions made by different techniques. *Policy2Code has the highest overall acceptance rate at 24%, followed by 15.2% for multi-class classification and 7.2% for CodeBERT, averaged over all assessments.*

[3]https://cwe.mitre.org/data/definitions/338.html
[4]https://cwe.mitre.org/data/definitions/336.html

TABLE III: Comparison of Policy2Code with Baselines on Compliance Assessment Tasks

Method	Classification Accuracy			Search MRR		
	CWE (Conceptual)	CWE	CBP	CWE (Conceptual)	CWE	CBP
CodeBERT	33.5	37.5	54.4	0.0223	0.0242	0.0217
Multi-class	42.82 ± 3.1	54.36 ± 1.9	60.04 ± 2.7	0.0297 ± 0.03	0.0312 ± 0.02	0.1310 ± 0.03
Policy2Code	**53.84** ± 6.1	**59.29** ± 2.0	**71.04** ± 2.8	**0.0462** ± 0.013	**0.0538** ± 0.01	**0.2182** ± 0.012

TABLE IV: Acceptance Rate in User Study (%)

Model	Compliant	Non-compliant	Overall
CodeBERT	9.68	6.38	7.2
Multi-class	27.01	11.57	15.2
Policy2Code	**41.93**	**17.64**	**24.13**

VII. Threats to Validity

As we formulate code compliance assessment as a machine learning problem, we see certain threats to the validity.

- ML models are known to carry biases from the training datasets. While more training schemes (pre-training as well as fine-tuning) can help improve model performance, some policies may still be under-represented in these training datasets. Certain issues may rarely occur in code and even rarely detected by human reviewers. The models trained on human code review comments and bug-fixes may not learn to detect these issues well.
- The benchmark dataset needs further curation. In using the CWE dataset, we see that certain policy descriptions are conceptual and loosely scripted that do not convey a strong recommendation or warning. Further, some ground truth labels are missing. Future work should further enhance the benchmark in both aspects to improve assessment.
- Finally, ML predictions may never meet 100% accuracy unlike carefully crafted static analysis rules. We expect automated tools to augment and not replace manual code reviews and static analyzer creation. Some reduced degree of manual vetting may still be required.

We need to understand how the performance of machine learning algorithms vary by data characteristics (policy and code) and how it can be improved by innovating on natural language (NL) and programming language (PL) research.

VIII. Conclusion

Code compliance assessment is an important problem in software development and an emerging area for machine learning. This paper explores novel research questions related to learning framework, training data, and evaluation setup. We proposed a representation learning approach that preserves the relationships between policies and their compliant, non-compliant, and irrelevant code examples via the vector distances between their embeddings. To overcome the lack of task-specific training data, we proposed repurposing general software datasets with pre-training, pre-fine-tuning, and filtering steps. We evaluated the impact of policy representations, losses, training schemes, and hyper-parameter optimization. Resulting Policy2Code model shows promising results for compliance classification and search tasks. Policy2Code achieves classification accuracies of (59%, 71%) and search MRR of (0.05, 0.21) on CWE and CBP, the two types of datasets in our benchmark. On the same datasets, CodeBERT baseline achieves classification accuracies of (37%, 54%) and MRR of (0.02, 0.02) respectively whereas multi-class baseline achieves classification accuracies of (54%, 60%) and MRR of (0.03, 0.13) respectively. In a user study of compliant and non-compliant findings, 24% detections from Policy2Code were accepted compared to only 7% detections from CodeBERT and 15% by the multi-class baseline.

We hope to encourage more theoretical and empirical research in automated code compliance assessment. We expect ML solutions to significantly reduce future manual efforts in code reviews and static analysis. Improvements can come from innovating on input transformations, use of program analysis and domain knowledge (API knowledge graph), and robust ML formulations.

Appendix A
Facet-masked Policy Representation

The idea for facet-masked representation is based on the conditional masking work, originally used in computer vision to compute image similarities across aspects such as color and texture [44]. The embedding of policy r is factorized depending on facet value y. The factorization is implemented via a conditional mask $\mathbf{m} \in R^{d \times n_k}$, where d is the embedding vector length and n_k is the number of facets (=2). The mask can be paramterized as $\mathbf{m} = \sigma(\beta)$, with σ denoting a rectified linear unit so that $\sigma(\beta) = max\{0, \beta\}$. The k^{th} column m_k plays the role of an element-wise gating function selecting the relevant subspace of the embedding dimensions to attend to the k^{th} facet. Additional loss regularization terms are used to encourage embeddings to be drawn from a unit ball (L_W) and for mask regularization (L_M).

Appendix B
Model Tuning Experiments

Performance of neural models can be significantly improved with proper hyper-parameter tuning (HPT). We used automatic selection of parameters (batch size, learning rate, etc) based on a validation dataset and experimented with triplet loss variations.

- *Batch All Triplet* loss computes the loss for all possible, valid triplets in the batch, i.e., anchor and positive must have the same label, anchor and negative a different label.
- *Batch Hard Triplet* loss computes the loss for all possible, valid triplets. It then looks for the hardest positive (largest $d(f_a, f_p)$) and the hardest negatives (smallest $d(f_a, f_n)$) per label class, and sums the loss only over them.

a) *Truncation errors occur when a primitive is cast to a primitive of a smaller size and data is lost in the conversion.* (CWE-197)

```
public static byte[] toByteArray(InputStream input, long
     size) throws IOException {
    if (size > Integer.MAX_VALUE) {
        throw new IllegalArgumentException(""Size greater
             than Integer max value: "" + size);
    }
    return toByteArray(input, (int) size);
}
```

b) *The software accepts XML from an untrusted source but does not validate the XML against the proper schema.* (CWE-112)

```
public static Document newDocument(String xmlString,
     boolean namespaceAware) throws SAXException,
     IOException, ParserConfigurationException {
    return XmlUtils.newDocument(new InputSource(new
         StringReader(xmlString)), namespaceAware);
}
```

c) *This program compares classes by name, which can cause it to use the wrong class when multiple classes can have the same name.* (CWE-486)

```
protected static String createTableRefName(final Object
     entity) {
    Class type = entity.getClass();
    type = (type == Class.class ? (Class) entity : type);
    return (type.getSimpleName() + '_');
}
```

d) *The product does not sufficiently enforce boundaries between the states of different sessions, causing data to be provided to, or used by, the wrong session.* (CWE-488)

```
protected void update(CollectionUpdateType force) throws
     IOException {
    State localState;
    synchronized (lock) {
        if (first) {
            state = checkState();
            state.last = System.currentTimeMillis();
            return;
        }
        localState = state.copy();
    }
    updateCollection(localState, force);
    localState.last = System.currentTimeMillis();
    synchronized (lock) {
        state = localState;
    }
}
```

e) *The product uses untrusted input when calculating or using an array index, but the product does not validate or incorrectly validates the index to ensure the index references a valid position within the array.* (CWE-129)

```
private static int calculateEndIndex( double[] array, int
     originalIndex ) {
    final int length = array.length;
    Exceptions.requireNonNull( array, ""array cannot be
         null"" );
    int index = originalIndex;
    if ( index < 0 ) {
        index = length + index;
    }
    if ( index < 0 ) {
        index = 0;
    }
    if ( index > length ) {
        index = length;
    }
    return index;
}
```

Fig. 3: *More Examples for* ***Compliant*** *Facet* - All examples are real detections from Policy2Code on CodeSearchNet-extended corpus. CWE ids are provided for reference only.

a) *Catching overly broad exceptions promotes complex error handling code that is more likely to contain security vulnerabilities.* (CWE-396)

```
private static byte[] decrypt(byte[] src, byte[] key)
     throws RuntimeException {
    try {
        SecureRandom sr = new SecureRandom();
        DESKeySpec dks = new DESKeySpec(key);
        SecretKeyFactory keyFactory = SecretKeyFactory.
             getInstance(DES);
        SecretKey securekey = keyFactory.generateSecret(dks
             );
        Cipher cipher = Cipher.getInstance(DES);
        cipher.init(Cipher.DECRYPT_MODE, securekey, sr);
        return cipher.doFinal(src);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

b) *Do not throw new checked exceptions (CheckedExceptions) forcing users to handle them elsewhere. Code should either propagate existing checked exceptions or handle them.*

```
public static Cipher newCipher(String algorithm) {
    try {
        return Cipher.getInstance(algorithm);
    }
    catch (NoSuchAlgorithmException e) {
        throw new IllegalArgumentException(""Not a valid
             encryption algorithm"", e);
    }
    catch (NoSuchPaddingException e) {
        throw new IllegalStateException(""Should not happen
             "", e);
    }
}
```

c) *Instead of repeatedly creating a new Random object to obtain multiple random numbers, create a single Random object and reuse it.*

```
private static final long SEED = 1234567890;
public int generateAccountID() {
    Random random = new Random(SEED);
    return random.nextInt();
}
```

d) *The product receives input that is expected to specify an index, position, or offset into an indexable resource such as a buffer or file, but it does not validate or incorrectly validates that the specified index/position/offset has the required properties.* (CWE-1285)

```
private static Object getCollectionProp(Object o, String
     propName, int index, String[] path) {
    o = _getFieldValuesFromCollectionOrArray(o, propName);
    if ( index + 1 == path.length ) {
        return o;
    } else {
        index++;
        return getCollectionProp(o, path[index], index,
             path);
    }
}
```

e) *The software does not properly handle when the expected number of parameters, fields,or arguments is not provided in input, or if those parameters are undefined.* (CWE-229)

```
public static <M extends Model> boolean isNew(M m, String
     pk_column) {
    final Object val = m.get(pk_column);
    return val == null || val instanceof Number && ((Number
         ) val).intValue() <= 0;
}
```

Fig. 4: *More Examples for* ***Non-compliant*** *Facet* - All examples are real detections from Policy2Code on CodeSearchNet-extended corpus. CWE ids are provided for reference only.

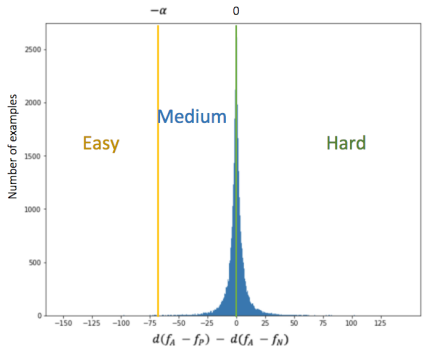

Fig. 5: Distribution of triplet distances can be used to tune margins and mine triplets.

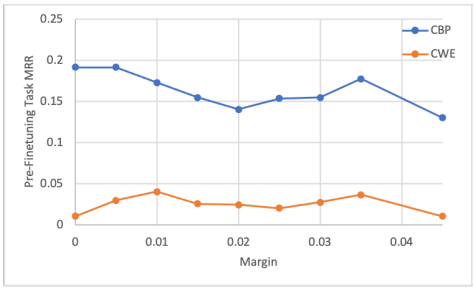

Fig. 6: Impact of margin on the pre-fine-tuning MRR of *Batch All Triplet* loss

- *Batch Semi-hard Triplet* loss computes the loss for all possible, valid triplets. It then looks for the semi hard positives and negatives and sums the loss only over them.
- *Batch Hard Triplet Soft-Margin* loss is a variation of the *Batch Hard* triplet loss where the loss over the hardest positive and the hardest negative examples are computed with a soft margin, e.g. $\log 1p(\exp(d(f_a, f_p) - d(f_a, f_n)))$.

The determination of hard, semi-hard, and easy triplets was made with a tunable margin parameter.

- Easy: $d(f_a - f_p) + \alpha < d(f_a - f_n)$. These are already well separated and not useful for model training.
- Medium: $d(f_a - f_p) < d(f_a - f_n) < d(f_a - f_p) + \alpha$. These triplets are typically more suited for optimisation.
- Hard: $d(f_a - f_n) < d(f_a - f_p)$. Selecting the hardest negative examples can sometimes lead to bad local minima and result in a collapsed model training [39].

Figure 5 shows the distribution of distances between the embeddings of text anchors and positive and negative code example for a sample of the dataset. The distribution is partitioned in three sections based on the margin, used to define easy, medium, and hard triplets. Figure 6 shows the effect of margin with various triplet modelling strategies. Table V compares the performance of loss variations on fine-tuning the CodeBERT model. The margin parameter is tuned for each variation except the soft-margin formulation which does not use a margin threshold. We observe that *Batch All Triplet loss has the best performance* and use it for all subsequent experiments.

TABLE V: Performance of triplet loss variations

Batch Triplet Loss Type	Overall Accuracy	Overall MRR
All Triplets	**57.2 ± 6.2**	**0.080 ± 0.02**
Hard Triplets	48.1 ± 4.7	0.063 ± 0.01
Semi-Hard Triplets	50.13 ± 2.8	0.068 ± 0.02
Hard Triplets Soft-Margin	53.12 ± 5.1	0.065 ± 0.04

REFERENCES

[1] Mitre Corporation, "The Common Weakness Enumeration (CWE) Initiative," http://cwe.mitre.org/, 2021, [Online; accessed 2021].

[2] Industry Research, "Global static code analysis software market report, history and forecast 2016-2027, breakdown data by companies, key regions, types and application," https://www.industryresearch.biz/global-static-code-analysis-software-market-18726250, p. 105, 2021, published: 2021-07-12.

[3] Amazon Web Services, *Amazon CodeGuru: Automate code reviews and optimize application performance with ML-powered recommendations*, 2021, https://aws.amazon.com/codeguru/.

[4] A. Habib and M. Pradel, "Neural bug finding: A study of opportunities and challenges," *CoRR*, vol. abs/1906.00307, 2019. [Online]. Available: http://arxiv.org/abs/1906.00307

[5] M. Pradel and K. Sen, "Deepbugs: A learning approach to name-based bug detection," *Proc. ACM Program. Lang.*, vol. 2, no. OOPSLA, Oct. 2018. [Online]. Available: https://doi.org/10.1145/3276517

[6] Y. Li, S. Wang, T. N. Nguyen, and S. Van Nguyen, "Improving bug detection via context-based code representation learning and attention-based neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, Oct. 2019.

[7] N. Stulova, A. Blasi, A. Gorla, and O. Nierstrasz, "Towards detecting inconsistent comments in java source code automatically," in *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2020, pp. 65–69.

[8] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, "Retrieval on source code: A neural code search," in *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, ser. MAPL 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 31–41. [Online]. Available: https://doi.org/10.1145/3211346.3211353

[9] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 964–974. [Online]. Available: https://doi.org/10.1145/3338906.3340458

[10] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 933–944. [Online]. Available: https://doi.org/10.1145/3180155.3180167

[11] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'08. Chicago, Illinois: AAAI Press, 2008, p. 646–651.

[12] H. Husain, H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *CoRR*, vol. abs/1909.09436, 2019. [Online]. Available: http://arxiv.org/abs/1909.09436

[13] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *International Conference on Learning Representations*. Vancouver, BC, Canada: OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=BJOFETxR-

[14] P. Rani, M. Birrer, S. Panichella, M. Ghafari, and O. Nierstrasz, "What do developers discuss about code comments?" in *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2021, pp. 153–164.

[15] J. K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, "CORE: automating review recommendation for code changes," *CoRR*, vol. abs/1912.09652, 2019. [Online]. Available: http://arxiv.org/abs/1912.09652

[16] T. H. M. Le, H. Chen, and M. A. Babar, "Deep learning for source code modeling and generation: Models, applications, and challenges," *ACM Comput. Surv.*, vol. 53, no. 3, Jun. 2020. [Online]. Available: https://doi.org/10.1145/3383458

[17] A. Naghshzan, L. Guerrouj, and O. Baysal, "Leveraging unsupervised learning to summarize apis discussed in stack overflow," in *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2021, pp. 142–152.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[19] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: https://www.aclweb.org/anthology/2020.findings-emnlp.139

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://www.aclweb.org/anthology/N19-1423

[21] P. Ramachandran, P. Liu, and Q. Le, "Unsupervised pretraining for sequence to sequence learning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 383–391. [Online]. Available: https://www.aclweb.org/anthology/D17-1039

[22] A. Aghajanyan, A. Gupta, A. Shrivastava, X. Chen, L. Zettlemoyer, and S. Gupta, "Muppet: Massive multi-task representations with pre-finetuning," 2021. Available: https://arxiv.org/abs/2101.11038

[23] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: Mining and searching internet-scale software repositories," *Data Min. Knowl. Discov.*, vol. 18, no. 2, p. 300–336, Apr. 2009. [Online]. Available: https://doi.org/10.1007/s10618-008-0118-x

[24] 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," in *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '15. Lincoln, Nebraska: IEEE Press, 2015, p. 260–270. [Online]. Available: https://doi.org/10.1109/ASE.2015.42

[25] C. Mcmillan, D. Poshyvanyk, M. Grechanik, Q. Xie, and C. Fu, "Portfolio: Searching for relevant functions and their usages in millions of lines of code," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, oct 2013. [Online]. Available: https://doi.org/10.1145/2522920.2522930

[26] M. Raghothaman, Y. Wei, and Y. Hamadi, "Swim: Synthesizing what i mean - code search and idiomatic snippet synthesis," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. Austin, TX, USA: IEEE, 2016, pp. 357–367.

[27] M. Lu, X. Sun, S. Wang, D. Lo, and Yucong Duan, "Query expansion via wordnet for effective code search," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, no. 22. Montreal, CA: IEEE, 2015, pp. 545–549.

[28] M. Allamanis, D. Tarlow, A. D. Gordon, and Y. Wei, "Bimodal modelling of source code and natural language," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. Lille, France: JMLR.org, 2015, p. 2123–2132.

[29] Y. Wan, J. Shu, Y. Sui, G. Xu, Z. Zhao, J. Wu, and P. S. Yu, "Multi-modal attention network learning for semantic source code retrieval," in *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '19, no. 34. San Diego, California: IEEE Press, 2019, p. 13–25. [Online]. Available: https://doi.org/10.1109/ASE.2019.00012

[30] X. Ling, L. Wu, S. Wang, G. Pan, T. Ma, F. Xu, A. X. Liu, C. Wu, and S. Ji, "Deep graph matching and searching for semantic code retrieval," *CoRR*, vol. abs/2010.12908, 2020. [Online]. Available: https://arxiv.org/abs/2010.12908

[31] J. P. Diniz, D. Cruz, F. Ferreira, C. Tavares, and E. Figueiredo, "Github label embeddings," in *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2020, pp. 249–253.

[32] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "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*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 964–974. [Online]. Available: https://doi.org/10.1145/3338906.3340458

[33] M. de Rezende Martins and M. A. Gerosa, "Concra: A convolutional neural networks code retrieval approach," in *Proceedings of the 34th Brazilian Symposium on Software Engineering*, ser. SBES '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 526–531. [Online]. Available: https://doi.org/10.1145/3422392.3422462

[34] Z. Yao, J. R. Peddamail, and H. Sun, "Coacor: Code annotation for code retrieval with reinforcement learning," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2203–2214. [Online]. Available: https://doi.org/10.1145/3308558.3313632

[35] M. Kaya and H. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, p. 1066, 2019.

[36] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, Dense, and Attentional Representations for Text Retrieval," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 329–345, 04 2021. [Online]. Available: https://doi.org/10.1162/tacl_a_00369

[37] D. Tunkelang, "Faceted search," in *Faceted Search*. NY, USA: Morgan and Claypool, 2009.

[38] O. Ben Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev, "Beyond basic faceted search," *WSDM'08 - Proceedings of the 2008 International Conference on Web Search and Data Mining*, vol. 8, pp. 33–44, 01 2008.

[39] R. Manmatha, C.-Y. Wu, A. Smola, and P. Krähenbühl, "Sampling matters in deep embedding learning," *2017 IEEE International Conference on Computer Vision (ICCV)*, vol. 1, no. 1, pp. 2859–2867, 2017.

[40] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, USA: IEEE Computer Society, 2015, pp. 815-823.
CoRR, vol. abs/1503.03832, 2015. [Online]. Available: http://arxiv.org/abs/1503.03832

[41] A. Hermans, L. Beyer, and B. Leibe, "In Defense of the Triplet Loss for Person Re-Identification," 2017. Available: https://arxiv.org/abs/2101.11038

[42] W. Chen, X. Chen, J. Zhang, and K. Huang, "Beyond triplet loss: a deep quadruplet network for person re-identification," *CoRR*, vol. abs/1704.01719, 2017. [Online]. Available: http://arxiv.org/abs/1704.01719

[43] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[44] A. Veit, S. Belongie, and T. Karaletsos, "Conditional similarity networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1. HI, USA: IEEE, 2017, pp. 1781–1789.

[45] N. Sawant and S. H. Sengamedu, "Learning-based identification of coding best practices from software documentation," in *2022 IEEE International Conference on Software Maintenance and Evolution*. Limassol, Cyprus: IEEE Computer Society, oct 2022, pp. 533-542.

[46] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. [Online]. Available: https://www.aclweb.org/anthology/P16-1162

[47] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.