Learning-based Identification of Coding Best Practices from Software Documentation

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

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

***Abstract*—Automatic identification of coding best practices can scale the development of code and application analyzers. We present *Doc2BP*, a deep learning tool to identify coding best practices in software documentation. Natural language descriptions are mapped to an informative embedding space, optimized under the dual objectives of binary and few shot classification. The binary objective powers *general classification* into known best practice categories using a deep learning classifier. The few shot objective facilitates *example-based classification* into novel categories by matching embeddings with user-provided examples at run-time, without having to retrain the underlying model. We analyze the effects of manually and synthetically labeled examples, context, and cross-domain information.**

We have applied *Doc2BP* to Java, Python, AWS Java SDK, and AWS CloudFormation documentations. With respect to prior works that primarily leverage keyword heuristics and our own *parts of speech* pattern baselines, we obtain 3-5% F1 score improvement for Java and Python, and 15-20% for AWS Java SDK and AWS CloudFormation. Experiments with four few shot use-cases show promising results (5-shot accuracy of 99%+ for Java *NullPointerException* and AWS Java *metrics*, 65% for AWS CloudFormation *numerics*, and 35% for Python best practices).

***Doc2BP* has contributed new rules and improved specifications in Amazon's code and application analyzers: (a) 500+ new checks in *cfn-lint*, an open-source AWS CloudFormation linter, (b) over 97% automated coverage of metrics APIs and related practices in *Amazon DevOps Guru*, (c) support for nullable AWS APIs in *Amazon CodeGuru*'s Java NullPointerException (NPE) detector, (d) 200+ new best practices for Java, Python, and respective AWS SDKs in *Amazon CodeGuru*, and (e) 2% reduction in false positives in *Amazon CodeGuru's* Java resource leak detector.**

***Index Terms*—natural language understanding, information extraction, embeddings, deep learning, few shot learning**

I. Introduction

Creating quality software requires an in depth knowledge of coding best practices on various aspects such as data structures, error handling, resource management, multiprocessing, and security. Coding best practices need to be *identified* before they can be incorporated in developer code or implemented as static analyzer checks. However, identification is non-trivial since best practice descriptions can be fragmented in documentation and hard to find due to significant differences in keywords, form, and semantics [1]–[3]. For example,

- *"Document.getText Method Now Allows for Partial Returns. For more efficient use, callers should invoke segment.setPartialReturn(true) and be prepared to receive a portion at a time"* (Java 11 Swing API reference)
- *"It is good programming practice to not use mutable objects as default values. Instead, use None as the default value and inside the function, check if the parameter is None and create a new list/ dictionary/ whatever if it is"* (Python 3.7 tutorial)
- *"When using the DynamoDBMapper to add or edit signed (or encrypted and signed) items, configure it to use a save behavior, such as PUT, that includes all attributes. Otherwise, you might not be able to decrypt your data"* (AWS Java SDK guide)
- *"The minimum maintenance window is 60 minutes"* (AWS CloudFormation user guide).

Our goal is to automate best practice identification from the documentation on various languages, frameworks, and applications to help scale the development of related code and application analyzers. Identified best practices can be implemented as new static analysis rules or used to enhance existing rules by covering more APIs and properties. Our primary use-case is *Amazon CodeGuru* (https://aws.amazon.com/codeguru/) [4], a developer tool that provides intelligent recommendations to improve code quality and identify an application's most expensive lines of code. The first three coding best practices described above were implemented as new rules in CodeGuru's Java, Python, and AWS Java SDK code analyzers, respectively. The fourth practice was used to update an existing rule in *cfn-lint*, a linter for AWS CloudFormation [5].

Prior works in automated best practice identification primarily rely on keyword heuristics curated on case-by-case basis. For example, extracting warnings and recommendations by matching keywords such as *'must'*, *'should'*, *'require'*, *'encourage'*, and *'recommend'* [2], [6], [7]. However, such heuristics fail to generalize for various reasons.

- *Keyword mismatch* - Keywords may differ across use-cases, for example in describing nullable APIs (*null* in Java and *None* in Python) or resources leaks (*terminate* or *kill* in Python and *tear_down* in AWS CloudFormation instead of *shutdown* and *close* in Java and AWS Java).
- *Context sensitivity* - Best practices may be contextual. For example, AWS SDK for Java [8] describes over 2000 *metrics* APIs to monitor the health and behavior of AWS services. The text is not consistently structured, requiring the context of each metric to be inferred before extracting

related best practice descriptions. For example, for Amazon Lex[1] *RuntimeSystemErrors*, relevant practices include *"The response code range for a system error is 500 to 599"*, *"Valid dimension for the PostContent operation with the Text or Speech InputMode: BotName, BotAlias, Operation, InputMode"*, and *"Unit: Count"*. For AWS Lambda[2] *Errors* metric, relevant practices include *"Sum statistic"*, *"To calculate the error rate, divide the value of Errors by the value of Invocations"*, etc.

- *Non-keyword patterns* - Best practice descriptions may not be keyword based. For example, AWS CloudFormation [9] describes *value constraints* on resources and properties of AWS cloud services such as a) *"TargetValue range is 8.515920e-109 to 1.174271e+108 (Base 10) or 2e-360 to 2e360 (Base 2)"*, (b) *"The minimum window is a 60 minute"*, (c) *"Up to five VPC security group IDs, of the form sg-xxxxxxxx"*, (d) *"The total number of allowed resources is 250"*. These practices are numeric patterns.

Our main contribution is *Doc2BP*, a deep learning tool to identify best practice descriptions from software documentation. The tool is aimed at reducing the overhead in maintaining multiple heuristics and simplifying new rule creation for different programming languages and frameworks. The tool supports two modes, *general classification* and *example-based classification*, powered by a common embedding space for natural language descriptions and jointly optimized under the dual objectives of binary and few shot classification respectively. The binary classification objective ensures coverage of known categories in available training data via a deep learning classifier, whereas the few shot objective allows classification into previously unseen categories based on the embedding similarity with a few user-labeled examples at inference time, without retraining the underlying deep learning model.

We have extensively applied *Doc2BP* on Java, Python, AWS Java SDK, and AWS CloudFormation documentations. The choice of documentations reflects the domains supported by Amazon CodeGuru at the time of writing this paper, and offers a good mix of general-purpose and specialized domains to study. Section III motivates the learning based approach using a case study on AWS CloudFormation. Sections IV and V present the representation learning formulation and overall Doc2BP system. Section VI covers extensive experiments with manually and synthetically labeled examples, context, and cross-domain information. With respect to prior keyword heuristics and our own *parts of speech* (POS) pattern baselines, we obtain 3-5% F1 score improvement in best practice detection for Java and Python and 15-20% for AWS Java SDK and AWS CloudFormation. We experiment with four use-cases in few shot setting with promising results (5-shot accuracy of 99%+ for Java *NullPointerException* and AWS Java *metrics*, 65% for AWS CloudFormation *numerics*, and 35% for Python best practices). These results indicate that *Doc2BP* is an effective solution for both general-purpose and specialized requirements. Section VII details real-world impact of *Doc2BP* on multiple code and application analyzers such as *cfn-lint* - an AWS CloudFormation linter [5], *Amazon DevOps Guru* - a cloud operations service to improve application availability [10], and *Amazon CodeGuru* - an automated code review tool for multiple programming languages and frameworks including Java, Python, and respective AWS SDKs [4].

[1]https://docs.aws.amazon.com/lex/latest/dg/monitoring-aws-lex-cloudwatch.html

[2]https://docs.aws.amazon.com/lambda/latest/dg/monitoring-metrics.html

II. Related Work

We now present prior work in extracting information from software documentation as well as related work in deep learning, natural language understanding, and few shot learning.

A. Information Extraction from Software Documentation

Monperrus et al. conducted a formal study of the types of knowledge in software documentation [6]. They proposed a list of keywords based on a manual review of Java documentation, RFC2119 - *"Keywords for use in RFCs to Indicate Requirement Levels"* [11], Oracle technical reports [12], and research papers [13]. Use-cases include extraction of method call practices (*"Subclasses should not call this internal method"*), subclassing practices (*"Subclasses may override any of the following methods: isLabelProperty, getImage, getText, dispose"*), or synchronization practices (*"If multiple threads access a hash map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally"*). This approach has been reused in other general-purpose studies [2], [3], [14]–[16] and extended for specialized requirements such as interrupt conditions [17] and performance concerns [7], [18]. For performance concerns, keywords can be *fast, slow, expensive, cheap, performance, speedup, efficient*, etc. and their inflections (e.g., *efficiency, efficiently)* [7], resulting in findings such as *"Raising this value decreases the number of seeds found, which makes mean_shift computationally cheaper"*. Table I lists popular prior work.

Few studies have used specialized natural language processing for healthcare [19], resource and method handling [20], [21], bug report analysis [22], [23] and software categorization [24]. A recent survey [25] indicates that less than 5% of research in security patterns uses natural language processing, for example, to extract access control requirements [26], [27], privacy policy visualization and summarization [28], inconsistent security requirements detection [29], and mining cyber threats from online documents [30], [31], and logs [32].

B. Related Work in Machine Learning

We now discuss concepts related to Doc2BP formulation.

1) Deep Learning and Natural Language Understanding: Deep learning has achieved a major breakthrough in many fields [33]–[36]. The seminal survey by Allamanis et al. [37] covers many applications such as code search, code completion, code generation, and documentation improvement. Subsequently many powerful neural models such as CodeBERT [38], PLBART [39], and CodeT5 [40] have been applied

TABLE I
KEYWORD HEURISTICS IN POPULAR SOFTWARE DOCUMENTATION MINING LITERATURE

Reference	Use-Case	Keyword Pattern
Monperrus et al. [6]	ControlFlow:Conditional	”(assum— only— debug— restrict— never— condition— strict—necessar— portab— strong)”
	ControlFlow:Temporal	”(call— invo— before — after — between — once — prior)”
	Recommend:Warning	”(warn—aware—error—note)”
	Recommend:Affirmative	”(must— mandat— require— shall— should— encourage— recommend— may)”
	Recommend:Alternative	”(desir—alternativ—addition)”
	Performance:Performance	”(performan—efficien—fast—quick—better—best)”
	Concurrency:Concurrency	”(concurren—synchron—lock—thread—simultaneous)”
	Subclassing:Subclassing	”(extend—overrid—overload—overwrit—re.?implement—sub.?class—super—inherit)”
Li et al. [2]	ControlFlow:Conditional	”(under the condition—whether— if —when—assume that)”
	ControlFlow:Temporal	”(before—after)”
	Recommend:Warning	”(insecure —susceptible — error— null— exception— susceptible— unavailable— not thread safe— illegal— inappropriate— insecure)”
	Recommend:Affirmative	”(must—should—have to—need to)”
	Recommend:Alternative	”(instead of—rather than—otherwise)”
	Recommend:Recommendation	”(deprecate—better to—best to—recommended—less desirable—discourage)”
	Recommend:Negative	”(do not—be not—never)”
	Recommend:Emphasis	”(none—only—always)”
	Recommend:Note	”(note that—notably—caution)”
Tao 2020 [7]	Performance:Performance	”(fast—slow—expensive—cheap—performan—speedup—computation—accelerat—intensi—scalable—efficien)”

to problems of bug detection [41], code review generation [42], and code and documentation synthesis [43], [44].

Detecting best practices, recommendations, and warnings is related to traditional natural language understanding tasks such as sentiment analysis [45]–[48] and suggestion mining [49]–[51]. In general literature, these tasks have been modeled using classical approaches such as parts of speech [52]–[54] and deep learning [35], [36], [55].

We have not seen any prior work generally applying deep learning or advanced natural language understanding for best practice identification from software documentation. Section 5.6 of the Allamanis survey [37] states *‘Also out-of-scope is work that combines natural language information with APIs’* and refers readers to investigate work already discussed above.

2) Few Shot Learning: Few-shot learning classifies new data having seen only a few training examples [56]. Few shot learning can be made tractable by incorporating in pre-training [57] knowledge from similar tasks, useful parameters, or data [58]–[60]. Similarity based algorithms such as matching networks [61] or prototypical networks [62] learn embeddings from training tasks that allow classification of unseen classes with few examples. Our approach is inspired by matching networks [61] and weakly supervised training [58].

III. FROM KEYWORDS TO LEARNING SYNTAX PATTERNS

We conducted a case study with AWS CloudFormation (a specialized framework for 200+ AWS services) to motivate learning based approaches for detecting non-keyword patterns given a few examples. Based on a manual documentation review of two AWS services - CloudTrail[3] and CodeCommit[4], we extracted about 50 best practice examples ranging from general recommendations (e.g, *“You can only use this property to add code when creating a repository with a AWS CloudFormation template at creation time”*; *“This property cannot be used for updating code to an existing repository”*), to alpha-numeric value constraints (e.g., *“Be between 3 and 128 characters”*; *“Start with a letter or number, and end with a letter or number”*). We chose *parts of speech* (POS) representations, mapping each word to its POS tag according to its syntactic role in the sentence (*noun, pronoun, adjective, determiner, verb*) [63]. We then applied PrefixSpan [64], a rule induction algorithm to infer frequent POS subsequence patterns. Given two sequences $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$, $\mathbf{x}$ is called a subsequence of $\mathbf{y}$, denoted as $\mathbf{x} \subseteq \mathbf{y}$ if there exist integers $1 \leq a_1 \leq a_2 \leq \ldots \leq a_m \leq n$ such that $x_1 \subseteq y_{a_1}$, $x_2 \subseteq y_{a_2}$, $x_m \subseteq y_{a_m}$. Figure 1 shows the frequent POS subsequence patterns learned from the selected examples. We observed the following:

[3]https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-cloudtrail-trail.html

[4]https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-codecommit-repository.html

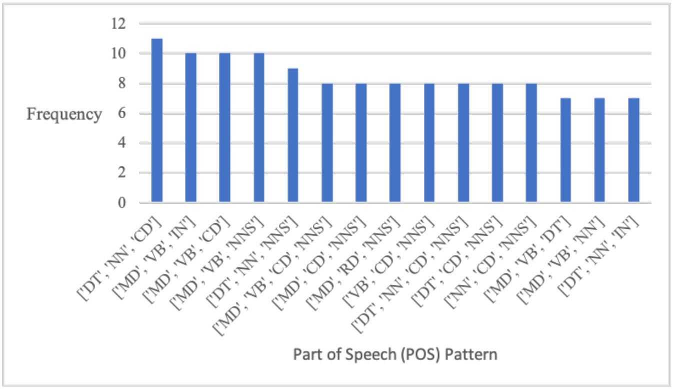


Fig. 1. POS patterns learned from documentation of two AWS services.

1) Ability to Replace Keyword-based Solutions: POS subsequences *[‘MD’, ‘VB’]* and *[‘MD’, ‘RB’]* occur in sentences whose POS sequence matches the following regular expression $\{[MD] >< .* > * < [VB]||[RB] >\}$. MD, VB, and RB are POS tags representing modal structure, base verbs and adverbs respectively. The pattern triggers on all imperative sentences, for example, *“The value must be no more than 255 characters”*. Comparing the detections from imperative POS pattern with keyword heuristics in Table I, we find a significant overlap as seen in Figure 2. *We find that the imperative pattern detects about 50% of all detections in the* affirmative *category and over 40% detections by the* conditional *category.*

2) Ability to Capture Non-Keyword Information: The subsequence *[‘DT’, ‘NN’, ‘CD’]* (in regular expression format

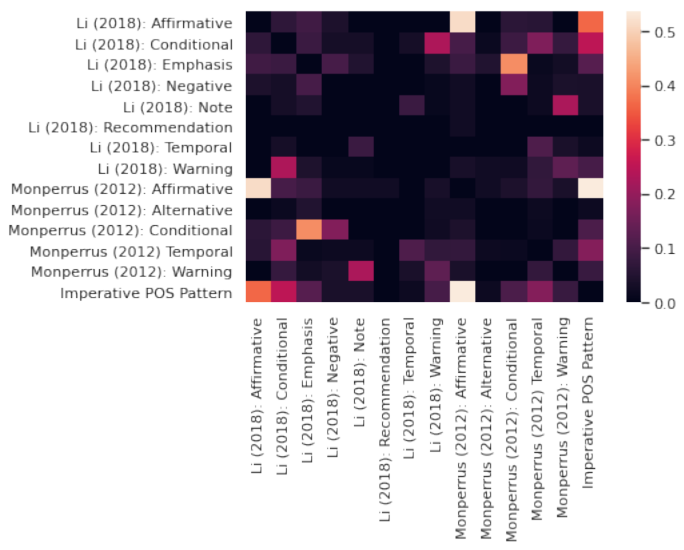


Fig. 2. Cooccurrence analysis between imperative POS pattern and multiple keyword heuristics shows that a single learned rule can significantly replace detections from multiple heuristics. The imperative pattern detects about 50% of all detections in the *affirmative* category and over 40% detections in the *conditional* category. Diagonal is suppressed to improve visual contrast.

$\{[DT] >< .* > * < [NN] >< .* > * < [CD] >\}$ is a pattern containing *CD*, e.g. cardinal digit. It matches a wide variety of numeric value constraints. For example, (a) *"The maximum length is 200 characters"*, (b) *"The number of resources cannot exceed 250 across events"*, (c) *"The count of allowed data resources is 250"*, and (d) *"This can be a number from 1 - 1024"*. The ability to detect the non-keyword patterns is an additional benefit of learning based approach.

To summarize, learning based algorithms can infer useful patterns from few examples, replace or augment keyword heuristics, and capture non-keyword requirements. This is possible because the natural language constructs as well as software documentation exhibit reasonably consistent structures. This insight has led to our detailed deep learning formulation, described below.

IV. Representation Learning Framework

We are given a training dataset S containing M labeled examples, $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$ where $\mathbf{x}_i \in \mathbb{R}^D$, is a D-dimensional feature vector and $y_i \in \{0, 1\}$ is a best practice label. For a subset S_c containing M_c known best practices e.g., $S_c \subset S = \{(\mathbf{x}_j, y_j) \mid y_j = 1\}_{j=1}^{M_c}$, we are also given an additional label $z_j \in \{1, \ldots, N\}$ to denote the category of best practice from N categories known at training time, for example, related to performance, security, subclassing, etc. For simplicity, we denote it as $S_c = \{(\mathbf{x}_j, z_j)\}_{j=1}^{M_c}$.

The core idea is to learn a metric space where each example can be encoded into a smaller L-dimensional dense representation (e.g., embedding) with function $f_\phi : \mathbb{R}^D \to \mathbb{R}^L, L \leq D$ and ϕ representing learnable embeddings. We optimize the embedding space under the dual objectives of binary and few shot classification. The binary classification objective ensures coverage of known categories in the available training data, by training a deep learning classifier for *general classification*. By default, such classifier cannot adapt to categories not included in the training set. To avoid model retraining for emerging requirements, we introduce a few shot learning capability that performs *example-based classification*, e.g. predicting new classes based on embedding similarity with few user-labeled examples at run time, without modifying model parameters. This objective encourages examples belonging to the same category to be co-located in the embedding space, thereby facilitating similarity based, non-parametric classification.

A. Binary Classification

Let g be any binary classifier parameterized via θ. If g is a logistic regression parameterized by $\theta = \{\mathbf{w}, b\}$, label y can be modeled as a function of input embedding $f_\phi(\mathbf{x})$ as follows:

$$\hat{y} = P(y = 1|\mathbf{x}; \phi, \theta) = \sigma(\mathbf{w}^\intercal f_\phi(\mathbf{x}) + b) \tag{1}$$

where $\sigma(a) = 1/(1 + e^{-a})$ is the logistic sigmoid function.

Loss between the predicted and actual probability distributions ($\hat{y}$ and y) is quantified using binary cross entropy (BCE).

$$L_{BCE} = -\sum_{i=1}^{M} y_i \log(\hat{y_i}) + (1 - y_i) \log(1 - \hat{y}_i) \tag{2}$$

B. Few Shot Classification

Our formulation of example-based classification is founded on two ideas. First, for the model to generalize to the test environment given a small number of new labeled examples (few shot), it should be trained under a similar setting. Secondly, the model should classify new test examples *without any changes to the model parameters*. For these purposes, we adopt the following *episodic training* strategy.

We create an n-way-k-shot episodic training, where the labeled dataset $S_c = \{(\mathbf{x}_j, z_j)\}_{j=1}^{M_c}$ is converted into several training episodes (e.g., mini-batches) by subsampling n training classes as well as k examples within each class. Each episode consists of $n \times k$ labeled examples (*support* set B) and an additional t examples (*test* set), also sampled from the same n classes. Test label z is modeled based on the embedding similarity of the test and the support examples. The similarity function between two embeddings, say $a(.,.)$ can be any attention kernel like a kernel density estimator or k-nearest neighbor that produces a similarity score. Similar to *matching network* [61], we model a as a soft-max over the cosine similarity $c(.,.)$ of embeddings, e.g.

$$a(\mathbf{x}, \mathbf{x'}; \phi) = \frac{e^{c(f_\phi(\mathbf{x}), f_\phi(\mathbf{x'}))}}{\sum_{\mathbf{x''}} e^{c(f_\phi(\mathbf{x}), f_\phi(\mathbf{x''}))}} \tag{3}$$

The label distribution $\hat{z}$ is a function of class similarities.

$$\hat{z} = P(z|\mathbf{x}; \phi) = \sum_{(x', z') \in B} a(\mathbf{x}, \mathbf{x'}; \phi) z' \tag{4}$$

Loss between the predicted and actual probability distributions ($\hat{z}$ and z) is quantified using general cross entropy (CE).

$$L_{CE} = -\sum_{i=1}^{M_c} z_i log(\hat{z}_i) \tag{5}$$

C. Overall Optimization

As the encoder function f_ϕ, we use BERT - *Bi-directional Encoder Representations from Transformers* [57]. The input to the BERT encoder (f) are natural language statements, each tokenized into word sequences and augmented with special tokens *[CLS]* and *[SEP]* to mark the start and the end of the statement. Context can be incorporated by augmenting the input representation with sentences preceding or following the candidate text. The input sequences are transformed into numeric sequences, followed by BERT processing to generate contextual embeddings. We use the average pooled output from BERT hidden states as the input embedding.

For binary classification, the embedding generated by BERT is passed through dropout and linear layers with soft-max output transformation. This is essentially the form of a feed-forward deep neural network for classification. Parameters ϕ and θ are optimized using stochastic gradient descent to minimize the combined loss $L = L_{BCE} + L_{CE}$. Figure 3 illustrates the combined machine learning architecture.

V. *Doc2BP* System Overview

Figure 4 presents an overview of *Doc2BP* with document parsing, best practice detection, and retriever modules.

The *Document Parsing* module loads diverse documentation formats and represents them as hypertext using BeautifulSoup [65]. Text is segmented based on line breaks and enclosing tag memberships (such as description lists and table rows) into semantically related components. Figure 5 shows a sample Python document, its HTML source, and the generated components that are subsequently input to best practice detection. Each component maintains a list of sentences, metadata (code tags, keywords, file name, headers, etc), and component relationships for navigating related best practices.

The *best practice identification* module supports keyword patterns (prior work), parts of speech (POS) patterns, and deep learning classifiers. The detectors can be used independently or collectively. Findings are labeled with detector metadata (e.g., *alphanumeric value, imperative, allowed pattern, security*) and stored in an index. In Figure 5(c), one best practice is detected from the first component - *"The named constructors are much faster than new() and should be preferred"*.

The *retriever* module supports user interaction. Users can input new test documentation and few-shot best practice examples, view findings, and provide feedback.

VI. Experiments

We experimented with four public software documentations: *Java 11 API Reference* [66], *Python 3.7 Reference* [67], *AWS Java SDK API Reference* [8], and *AWS CloudFormation User Guide* [9]. The choices reflect the domains supported by Amazon CodeGuru at the time of writing this paper, and offers a good mix of general and specialized domains to study.

Datasets: Deep learning models are data hungry. We synthesized a *weakly labeled* dataset for pre-training and a small manually curated ground truth for fine-tuning and evaluation. The pre-training dataset is created by applying the keyword heuristics in Table I and the POS patterns in Section III. Sentences detected by these techniques are labeled as positive $(y = 1)$. A subset of sentences where none of the detectors triggered are labeled as negative. We expect the labels to have some noise, hence termed *weakly labeled*. To simplify few shot training, examples that mapped to multiple categories in a batch were removed. We curated the ground truth by manually labeling 1000 sentences per documentation, split 70%-30% for fine-tuning and evaluation. Table II shows dataset properties.

TABLE II
Documentation Corpus used in Experiments

Documentation Corpus	Size (MB)	Documents	Sentences	#Positives in 1K Ground Truth
Java [66]	97	13,386	537,277	451
Python [67]	13	509	71,928	698
AWS Java SDK [8]	177	31,355	820,001	441
AWS CloudFormation [9]	2	93	14,851	632

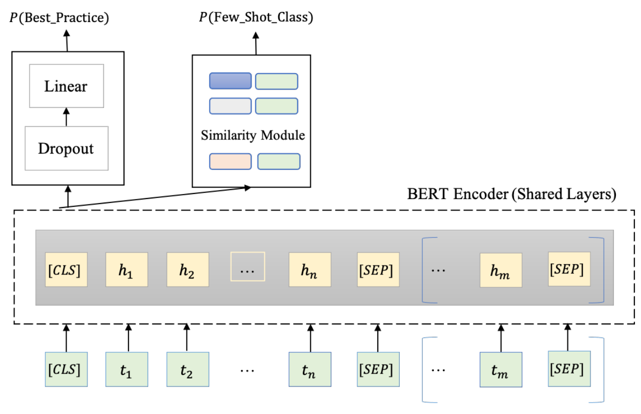
Model Training: We used HuggingFace implementation of BERT (*BERT-base-case*) previously pre-trained on *BookCorpus*, a dataset consisting of 11,038 unpublished books and English Wikipedia. We further pre-trained and fine-tuned BERT using above datasets, and experimented with the following:

- *Incorporating context*: We used *non-contextual* and *contextual* representations. For context, we further evaluated the difference between using preceding context (preceding sentences) and surrounding context (both preceding and following sentences) in the paragraph.
- *Combining training corpora*: We used three schemes (a) *Self* - Each documentation corpus is maintained separate and used to train language-specific models; (b) *Java + Python* - A combined corpus of general purpose programming language documentation, and (c) *All* - A combined corpus of all four documentation types. This helps evaluate if patterns are transferable across domains.

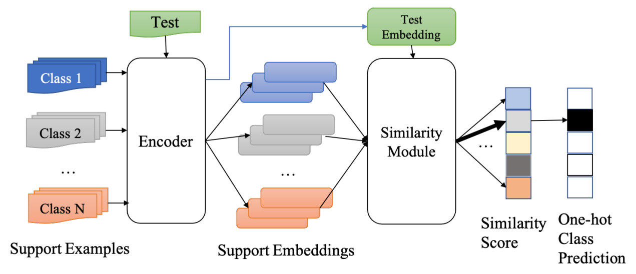
All experiments were run on a p3.16xlarge EC2 machine with hyper-parameter tuning (sequence length, the number of training shots, batch size, and learning rates). Since deep learning optimization can be sensitive to initialization, experiments were replicated with 10 random seeds. For few shot fine-tuning, we achieve the best validation performance with *N=4*, *K=3* episodes for 128-dimensional representations.

A. Pre-training Results

Table III shows pre-training results. We observe that *pre-training with the weakly labeled data is effective by itself.* Without using any ground truth labels, we can achieve accuracy of 64% for AWS Java SDK, 67% for AWS CloudFormation, 54% for Java, and 73% for Python. Secondly, *combining training data across documentation provides impressive gains for specialized frameworks - AWS Java SDK (+7%) and AWS CloudFormation (+20%) and small but significant gains for Java (+1.5%) and Python (+2.5%) respectively.* Finally, *contextual representations lead to better performance than non-contextual representations when using combined training*


P(Best_Practice)
P(Few_Shot_Class)
Linear
Dropout
Similarity Module
BERT Encoder (Shared Layers)
[CLS] h1 h2 … hn [SEP] … hm [SEP]
[CLS] t1 t2 … tn [SEP] … tm [SEP]

(a) BERT-based representation learning under dual objectives



(b) Few shot classification for example-based identification

Fig. 3. *Doc2BP* representation learning and inference framework

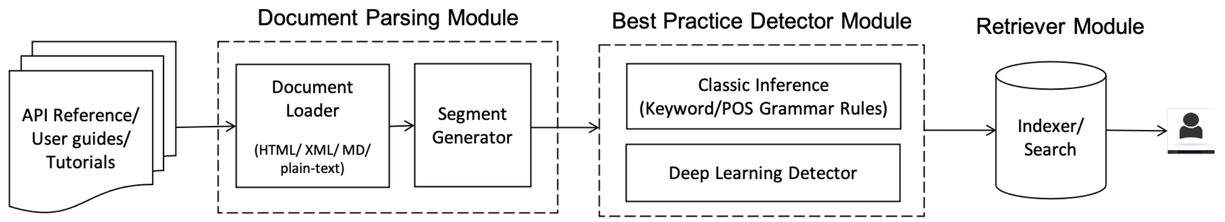

Document Parsing Module
Best Practice Detector Module
Retriever Module
API Reference/ User guides/ Tutorials
Document Loader
(HTML/ XML/ MD/ plain-text)
Segment Generator
Classic Inference (Keyword/POS Grammar Rules)
Deep Learning Detector
Indexer/ Search

Fig. 4. *Doc2BP* System Overview

corpora. For AWS *All* strategy, about 20% accuracy gain is observed, with others at a moderate gain of 5-8%.

B. Fine-tuning Results

We next experimented with fine-tuning the encoder by adding small amounts of ground truth examples. We sampled 175, 350, and 525 (corresponding to 25%, 50%, and 75%) examples from the ground truth training partition and tuned the performance for 175 validation examples. From the fine-tuning performance in Table IV and comparing it with the pre-training performance in Table III, we observe the following: (a) *CloudFormation records a significant improvement (67% in pre-training versus 83% for fine-tuning) with just 175 examples*. This may be attributed to the high regularity of its documentation and best practice descriptions; (b) *Fine-tuning using domain-specific data helps specialized languages as noted by the best performance on AWS Java SDK and Cloud-Formation from the "Self" training mode*; (c) *For general purpose programming languages, fine-tuning with combined corpora leads to slightly better results*.

C. Comparison with Baselines

Table V compares the best practice classification performance of *Doc2BP* with various strategies: (a) popular keyword heuristics shown in Table I by Li et al. [2] and Monperrus et al. [6] and (b) imperative and numeric *parts of speech* (POS) patterns from Section III. For better visualization of this large table, we have bold-faced highest values in each column. We observe that (a) Doc2BP *significantly out-performs other baselines in recall and F1 classification metrics, with F1-score improvement of 15-20% for AWS Java SDK and CloudFormation and 3-5% for Java and Python*. Doc2BP *captures over 75% of ground truth examples across all cases*. (b) Some keyword detectors (*Recommendation*, *Note*, and *Affirmative*) are extremely precise, but also narrow e.g., having a very small recall, hence also a small F1 score. Some Java-specific curated keyword patterns (*Alternative* and *Negative*) have lower generalization on specialized frameworks. (c) *The recall and F1 of POS patterns are comparable to the collection of multiple curated keywords in case of AWS Java SDK and CloudFormation*. They under-perform for the more general use-cases of Java and Python. Gaps are expected to reduce if more POS patterns are included, e.g. from Figure 1.

Doc2BP assigns a probabilistic score to each detection, helping generate a ranked list of best practices for a more efficient manual review compared to other non-probabilistic baselines. Figure 6 shows the ROC curves and the area under the curve (AUC) for *Doc2BP* on different documentations.

D. Few Shot: Example based identification

We tested four increasingly complex scenarios from keyword contexts, parts of speech, to higher order semantics.

- Null Pointer Exception (NPE) - An important use-case in Java is to determine if an API can return null values. Nullable APIs can be detected from documentation using *'null'* keyword. For example, *S3 getCachedResponseMetadata* API is described to *"return the response metadata for the specified request, or null if*

TABLE III
PRE-TRAINING EXPERIMENTS WITH CROSS-DOMAIN INFORMATION AND CONTEXT: RESULTS ON GROUND TRUTH TEST DATA

Pre-training Data	**Input Format**	**AWS Java SDK**	**AWS CloudFormation**	**Java**	**Python**
Self	Non-contextual	57.4% ± 0.6%	46.7% ± 0.4%	50.5% ± 0.3%	70.8% ± 0.1%
Self	Preceding context	44.1% ± 0.9%	46.1% ± 0.8%	52.3% ± 0.2%	65.6% ± 0.3%
Self	Surrounding context	44.0% ± 0.5%	45.2% ± 0.2%	47.0% ± 0.3%	68.1% ± 0.1%
Java + Python	Non-contextual	51.5% ± 0.8%	59.5% ± 0.6%	50.6% ± 0.2%	66.3% ± 0.5%
Java + Python	Preceding context	63.5% ± 0.9%	67.4% ± 0.3%	54.2% ± 0.4%	72.9% ± 0.5%
Java + Python	Surrounding context	63.4% ± 0.2%	67.5% ± 0.0%	53.8% ± 0.3%	73.1% ± 0.2%
All	Non-contextual	45.6% ± 0.2%	62.2% ± 0.2%	45.3% ± 0.2%	68.8% ± 0.1%
All	Preceding context	57.9% ± 0.8%	63.0% ± 0.4%	49.5% ± 0.4%	67.5% ± 0.5%
All	Surrounding context	64.1% ± 0.0%	67.2% ± 0.3%	53.2% ± 0.3%	73.2% ± 0.7%

TABLE IV
FINE-TUNING EXPERIMENTS WITH CROSS-DOMAIN INFORMATION AND TRAINING SIZE: RESULTS ON GROUND TRUTH TEST DATA

Finetuning Data	**Training Size**	**AWS Java SDK**	**AWS CloudFormation**	**Java**	**Python**
Self	175	69.6% ± 0.05%	83.0% ± 0.01%	73.4% ± 0.08%	53.2% ± 0.05%
Self	350	76.9% ± 0.09%	82.3% ± 0.08%	73.4% ± 0.09%	68.8% ± 0.11%
Self	525	78.9% ± 0.11%	83.3% ± 0.06%	82.4% ± 0.02%	72.9% ± 0.18%
Java + Python	175	69.2% ± 0.06%	66.6% ± 0.14%	76.7% ± 0.10%	70.2% ± 0.20%
Java + Python	350	72.9% ± 0.07%	72.8% ± 0.16%	77.7% ± 0.12%	72.2% ± 0.19%
Java + Python	525	71.2% ± 0.07%	67.5% ± 0.15%	84.4% ± 0.11%	74.9% ± 0.19%
All	175	71.6% ± 0.07%	67.5% ± 0.15%	68.1% ± 0.06%	57.6% ± 0.13%
All	350	77.9% ± 0.08%	77.0% ± 0.14%	77.7% ± 0.09%	71.2% ± 0.15%
All	525	78.9% ± 0.10%	79.0% ± 0.20%	83.1% ± 0.11%	72.9% ± 0.18%

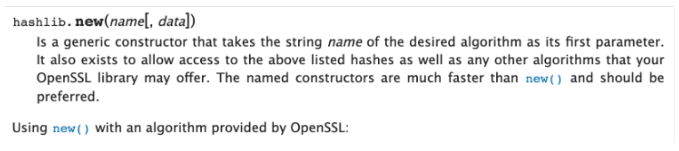
TABLE V
COMPARING BEST PRACTICE DETECTION PERFORMANCE OF BASELINES ON GROUND TRUTH TEST DATA

Detector	**Precision**				**Recall**				**F1**			
	AWS Java	CloudFormation	Java	Python	AWS Java	CloudFormation	Java	Python	AWS Java	CloudFormation	Java	Python
Li et al. [2]: Affirmative	84.0%	86.2%	73.7%	93.3%	15.9%	25.9%	10.3%	13.6%	26.8%	39.8%	18.1%	23.7%
Li et al. [2]: Alternative	25.0%	**100.0%**	60.0%	94.1%	0.8%	1.6%	2.2%	7.8%	1.5%	3.1%	4.3%	14.3%
Li et al. [2]: Conditional	46.2%	52.3%	66.1%	94.1%	13.6%	11.9%	30.1%	15.5%	21.1%	19.4%	4.1%	2.7%
Li et al. [2]: Emphasis	68.8%	78.8%	56.3%	79.2%	8.3%	13.5%	6.6%	9.2%	1.5%	2.3%	1.2%	1.7%
Li et al. [2]: Negative	66.7%	20.0%	80.0%	75.0%	3.0%	0.5%	2.9%	1.5%	5.8%	1.0%	5.7%	2.9%
Li et al. [2]: Note	0.0%	**100.0%**	**100.0%**	**100.0%**	0.0%	0.5%	3.7%	1.9%	0.0%	1.0%	7.1%	3.8%
Li et al. [2]: Performance	33.3%	0.0%	0.0%	100.0%	2.3%	0.0%	0.0%	1.9%	4.3%	0.0%	0.0%	3.8%
Li et al. [2]: Recommendation	**100.0%**	**100.0%**	50.0%	**100.0%**	0.8%	0.5%	1.5%	3.9%	1.5%	1.0%	2.9%	7.5%
Li et al. [2]: Temporal	61.5%	35.3%	57.1%	84.6%	6.1%	3.1%	2.9%	5.3%	11.0%	5.7%	5.6%	1.0%
Li et al. [2]: Warning	45.5%	50.0%	53.7%	64.7%	3.8%	1.6%	32.4%	10.7%	7.0%	3.0%	40.4%	18.3%
Li et al. [2]: All	58.1%	66.7%	61.2%	85.5%	55.0%	59.0%	**82.0%**	71.0%	56.3%	62.6%	69.9%	77.8%
Monperrus et al. [6]: Affirmative	76.5%	84.4%	69.4%	92.5%	19.7%	33.7%	18.4%	18.0%	31.3%	4.8%	29.1%	30.1%
Monperrus et al. [6]: Alternative	33.3%	20.0%	**100.0%**	83.3%	3.0%	0.5%	3.7%	2.4%	5.6%	1.0%	7.1%	4.7%
Monperrus et al. [6]: Concurrency	28.6%	30.8%	50.0%	70.0%	1.5%	2.1%	8.8%	3.4%	2.9%	3.9%	15.0%	6.5%
Monperrus et al. [6]: Conditional	59.1%	78.4%	57.1%	82.4%	9.8%	15.0%	5.9%	6.8%	16.9%	25.2%	10.7%	12.6%
Monperrus et al. [6]: Performance	44.4%	50.0%	**100.0%**	**100.0%**	3.0%	0.5%	0.7%	1.9%	5.7%	1.0%	1.5%	3.8%
Monperrus et al. [6]: Subclassing	25.0%	75.0%	24.0%	76.2%	0.8%	1.6%	4.4%	7.8%	1.5%	3.0%	7.5%	14.1%
Monperrus et al. [6]: Temporal	38.6%	40.0%	52.8%	62.8%	12.9%	6.2%	20.6%	13.1%	19.3%	10.8%	29.6%	21.7%
Monperrus et al. [6]: Warning	63.6%	62.5%	61.9%	69.2%	10.6%	2.6%	9.6%	13.1%	18.2%	5.0%	16.6%	22.0%
Monperrus et al. [6]: All	52.6%	68.2%	54.7%	76.1%	61.0%	62.0%	72.0%	67.0%	56.6%	65.0%	62.2%	71.0%
Imperative POS Patterns (Ours)	66.7%	76.1%	63.6%	81.0%	53.0%	56.0%	36.0%	39.3%	59.1%	64.5%	46.0%	52.9%
Numeric POS Patterns (Ours)	31.4%	65.6%	44.4%	60.0%	24.2%	30.6%	23.5%	26.2%	27.4%	41.7%	30.8%	36.5%
Doc2BP (Ours)	64.1%	76.1%	66.2%	87.2%	**81.1%**	**89.1%**	76.5%	**76.2%**	**71.6%**	**82.1%**	**71.0%**	**81.3%**

none is available". However, keyword patterns can also falsely trigger on null-safe APIs. For example, the *DynamoDBV2 GetItemOutcome* API is described to *"return the (non-null) result of item retrieval"*. We use nullable API descriptions as a new user-defined class and include null-safe sentences into a generic negative class.

- AWS Java Metrics - We use the AWS Java *metrics* contexts as a new user-defined class. For example, (a) *"Use with the Sum statistic to measure throughput and with the Samples statistic to measure IOPS"*; (b) *"moving to a larger instance type when this value reaches 85 percent"*. These descriptions are likely to use measurement related terms such as *statistic, quantity, minimum, maximum.*
- AWS CloudFormation numerics - We use the CloudFormation value constraints as another user-defined class. As noted earlier, numeric constraints map to part of speech patterns (using POS tag *CD* e.g., cardinal digit), a higher-order linguistic feature compared to keywords. We expect this category to be more complex than the previous two.
- Python best practices - Finally, we create a class for Python best practice considerations such as *"do not use mutable objects as default values"*, or *"Use datetime instead of time for local clocks"*. These sentences are the most diverse and semantic sensitive, thus most difficult to learn from a similar few shot setting.

We created a 5-way-K-shot dataset using 20 examples from above classes. For 10 rounds each with a different random seed, we sampled K examples to be the support set and the remaining examples as the test set. From the results in Figure VI, we observe that: (a) *Java NPE and AWS Java metric classes with local, keyword-type patterns enjoy high performance with a few examples* (over 95% accuracy for Java NPE class and

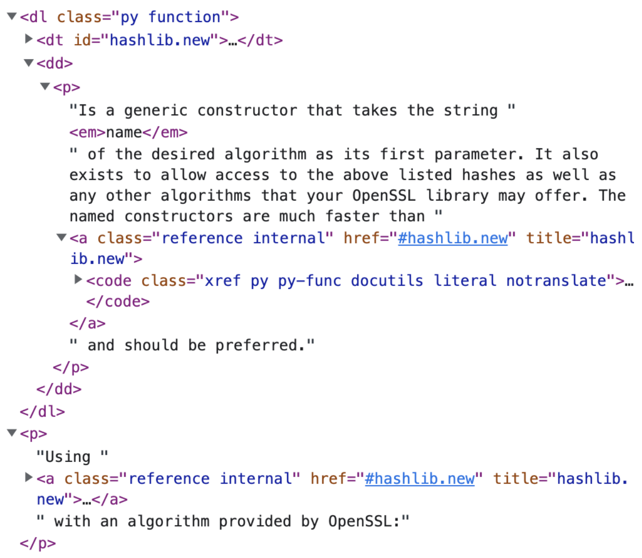


hashlib.**new**(*name*[, *data*])

Is a generic constructor that takes the string *name* of the desired algorithm as its first parameter. It also exists to allow access to the above listed hashes as well as any other algorithms that your OpenSSL library may offer. The named constructors are much faster than `new()` and should be preferred.

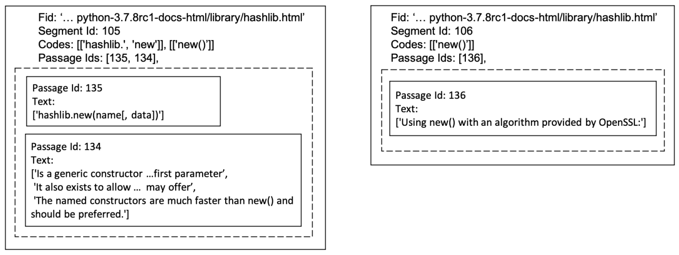
Using `new()` with an algorithm provided by OpenSSL:

(a) Python document snippet



```
▼<dl class="py function">
  ▶<dt id="hashlib.new">…</dt>
  ▼<dd>
    ▼<p>
       "Is a generic constructor that takes the string "
       <em>name</em>
       " of the desired algorithm as its first parameter. It also
       exists to allow access to the above listed hashes as well as
       any other algorithms that your OpenSSL library may offer. The
       named constructors are much faster than "
      ▼<a class="reference internal" href="#hashlib.new" title="hashl
       ib.new">
        ▶<code class="xref py py-func docutils literal notranslate">…
         </code>
       </a>
       " and should be preferred."
     </p>
   </dd>
 </dl>
▼<p>
   "Using "
  ▶<a class="reference internal" href="#hashlib.new" title="hashlib.
   new">…</a>
   " with an algorithm provided by OpenSSL:"
 </p>
```

(b) HTML Source of the input snippet



Fid: '… python-3.7.8rc1-docs-html/library/hashlib.html'
Segment Id: 105
Codes: [['hashlib.', 'new']], [['new()']]
Passage Ids: [135, 134],

Passage Id: 135
Text:
['hashlib.new(name[, data])']

Passage Id: 134
Text:
['Is a generic constructor …first parameter',
'It also exists to allow … may offer',
'The named constructors are much faster than new() and should be preferred.']

Fid: '… python-3.7.8rc1-docs-html/library/hashlib.html'
Segment Id: 106
Codes: [['new()']]
Passage Ids: [136],

Passage Id: 136
Text:
['Using new() with an algorithm provided by OpenSSL:']

(c) Document Segmentation

Fig. 5. When a document is input to the *Document Parsing* module, its HTML source is analyzed and segmented into cohesive components with metadata.

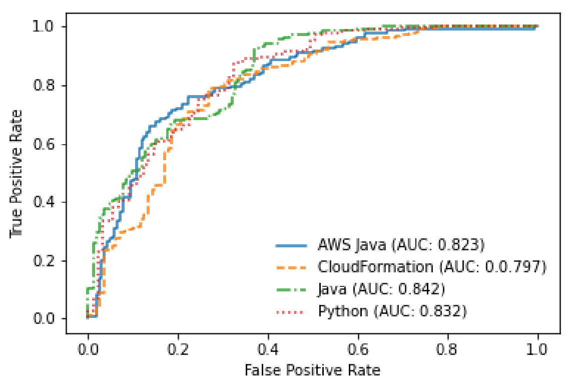


Fig. 6. *Doc2BP* ROC Curves and AUC by Documentation Type

TABLE VI
K-SHOT RETRIEVAL ACCURACY

Task	K=1	K=3	K=5
Java Null Pointer Exception	95.5% ± 0.8%	99.3.% ± 0.7%	99.8% ± 0.4%
AWS Java Metrics	90.0% ± 1.2%	98.0% ± 1.1%	99.5% ± 0.9%
CloudFormation Numerics	55.0% ± 1.2%	60.0% ± 0.7%	65.0% ± 0.8%
Python Best Practices	15.0% ± 2.3%	20.0% ± 1.8%	35.0% ± 1.9%

over 90% performance for AWS Java Metrics). (b) *Classes with more variations have low to moderate performance in 1-shot setting and it can be subsequently improved by providing more examples*, (e.g. from 55% to 65% for CloudFormation numerics and from 15% to 35% or Python best practices).

VII. Real-World Applications

Doc2BP has powered several real-world use-cases such as:

- *Improving Coverage of Existing Analyzers*:
  - The *cfn-lint* [5] is an open-source linter for AWS CloudFormation. Mining from documentation, we contributed 500+ new rules in various categories (number of rules in bracket): allowed patterns (158), string constraints (104), numeric range (89), valid values (83), non-compatible value pairs (32), single-ton constraints (21), and container limits (23).
  - *Amazon DevOps Guru* [10] is an ML-powered cloud operations service, tracking metrics related to different AWS services. At the time of writing, metrics extraction had been automated with *Doc2BP* covering over 97%+ metrics of over 200+ services.
  - We also identified nullable APIs across all AWS services to support AWS APIs in *Amazon CodeGuru*'s Java NullPointerException (NPE) detector. This is a customized version of Infer's *biabduction* detector.
- *Prioritizing New Analyzers*: We sourced hundreds of rules for Java, Python, and their AWS SDKs, and implemented 200+ new rules in *Amazon CodeGuru*. These rules cover standard library, scientific operations, security, resource management, concurrency, error handling, logging, etc.
- *Reducing False Positives*: In Java, objects of classes that implement *Closeable* or *AutoCloseable* interface need to be closed after use, with certain exceptions such as *java.io.ByteArrayInputStream* whose *close()* method is a no-op. Flagging missing *close()* operations on such APIs will be considered as false positives. By identifying such resources from description (e.g., containing phrase *'close has no effect'*), we reduced the false positives in CodeGuru Java resource leak detector [68] by 2%.

VIII. Conclusion and Future Work

We present *Doc2BP*, a novel deep learning tool to identify coding best practice descriptions in software documentation. It supports *general classification* to identify best practices of known categories and an *example-based classification* to handle novel categories by matching embeddings with user-provided examples at run-time, without retraining the underlying model. The core idea is to train an informative embedding space for natural language descriptions under the dual objectives of binary and few shot classification.

We have extensively applied *Doc2BP* on Java, Python, AWS Java SDK, and AWS CloudFormation documentations. We have analyzed the effects of manually and synthetically labeled examples, and incorporating context and cross-domain information. The detection performance significantly outperforms known keyword heuristics as well as parts of speech pattern baselines. *Doc2BP* obtains 3-5% F1 score improvement in best practice detection for Java and Python and 15-20% for AWS Java SDK and AWS CloudFormation. Our experiments with four few shot use-cases show promising results (5-shot accuracy of 99%+ for Java *NullPointerException* and AWS Java *metrics*, 65% for AWS CloudFormation *numerics*, and 35% for Python best practices). *Doc2BP* is used in Amazon and has successfully contributed to many code and application analyzers including *Amazon CodeGuru*.

We hope our contributions will advance the state of the art in automated software engineering. We continue to explore related research questions such as *How to handle obsolete documentations and best practices? How to extract non-contiguous or non-local metadata for a best practice? How to fuse knowledge from multiple descriptions of the same best practice? How to detect and resolve conflicting or ambiguous descriptions?* In the near term, we do expect some level of manual curation to address above questions. We believe further research in representations and modeling architectures can help especially for emerging and unfamiliar domains.

IX. Acknowledgement

Many colleagues have contributed to reviewing findings and their integration with various analyzers: Kevin DeJong, Anton Emelyanov, Ran Fu, Pranav Garg, Rajdeep Mukherjee, Thomas Cottenier, Ben Liblit, Omer Tripp, and Rahul Tongia.

References

[1] A. Marques, N. C. Bradley, and G. C. Murphy, "Characterizing task-relevant information in natural language software artifacts," in *2020 IEEE International Conference on Software Maintenance and Evolution*, 2020, pp. 476–487.

[2] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu, and X. Zhao, "Improving api caveats accessibility by mining api caveats knowledge graph," in *2018 IEEE International Conference on Software Maintenance and Evolution*, 2018, pp. 183–193.

[3] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

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

[5] K. DeJong, *AWS CloudFormation Linter*, 2020, https://github.com/aws-cloudformation/cfn-python-lint.

[6] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, "What should developers be aware of? an empirical study on the directives of api documentation," *Empirical Software Engineering*, vol. 17, no. 6, p. 703–737, Dec. 2012.

[7] Y. Tao, J. Jiang, Y. Liu, Z. Xu, and S. Qin, "Understanding performance concerns in the api documentation of data science libraries," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, New York, NY, USA: Association for Computing Machinery, 2020, p. 895–906.

[8] Amazon Web Services, *AWS SDK for Java API Reference - 1.12.86*, 2021, https://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/.

[9] Amazon Web Services, *AWS CloudFormation User Guide*, 2021, https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html, Accessed 2020.

[10] Amazon Web Services, *Amazon DevOps Guru: ML-powered cloud operations service to improve application availability*, 2021, https://aws.amazon.com/devops-guru/.

[11] S. O. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, RFC 2119, Mar. 1997, https://rfc-editor.org/rfc/rfc2119.txt.

[12] K. Smith and D. Kramer, *Requirements for writing java API specifications*, January 2003, https://www.oracle.com/java/technologies/javase/api-specifications.html.

[13] U. Dekel and J. D. Herbsleb, "Improving api documentation usability with knowledge pushing," in *2009 IEEE 31st International Conference on Software Engineering*, 2009, pp. 320–330.

[14] X. Zhao, Z. Xing, M. A. Kabir, N. Sawada, J. Li, and S.-W. Lin, "Hdskg: Harvesting domain specific knowledge graph from content of webpages," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 56–67.

[15] X. Ren, J. Sun, Z. Xing, X. Xia, and J. Sun, "Demystify official api usage directives with crowdsourced api misuse scenarios, erroneous code examples and patches," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, New York, USA: Association for Computing Machinery, 2020, p. 925–936.

[16] M. P. Robillard and Y. B. Chhetri, "Recommending reference api documentation," *Empirical Software Engineering*, vol. 20, no. 6, p. 1558–1586, Dec. 2015.

[17] L. Tan, Y. Zhou, and Y. Padioleau, "acomment: mining annotations from comments and code to detect interrupt related concurrency bugs," in *Proc. 33rd International Conference on Software Engineering*, 2011, pp. 11–20.

[18] M. Selakovic and M. Pradel, "Performance issues and optimizations in javascript: An empirical study," in *2016 IEEE/ACM 38th International Conference on Software Engineering*, 2016, pp. 61–72.

[19] S. Ezami, "Extracting non-functional requirements from unstructured text," https://uwspace.uwaterloo.ca/handle/10012/12889, 2018.

[20] H. Zhong, L. Zhang, T. Xie, and H. Mei, "Inferring specifications for resources from natural language api documentation," *Automated Software Engineering*, vol. 18, no. 03, pp. 227–261, may 2011.

[21] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar, "Inferring method specifications from natural language api descriptions," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE 12. IEEE Press, 2012, p. 815–825.

[22] P. Mazrae, M. Izadi, and A. Heydarnoori, "Automated recovery of issue-commit links leveraging both textual and non-textual data," in *2021 IEEE International Conference on Software Maintenance and Evolution*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 263–273.

[23] C. Mills, J. Pantiuchina, E. Parra, G. Bavota, and S. Haiduc, "Are bug reports enough for text retrieval-based bug localization?" in *2018 IEEE International Conference on Software Maintenance and Evolution*, 2018, pp. 381–392.

[24] A. LeClair, Z. Eberhart, and C. McMillan, "Adapting neural text classification for improved software categorization," in *2018 IEEE International Conference on Software Maintenance and Evolution*, 2018, pp. 461–472.

[25] H. Washizaki, T. Xia, N. Kamata, Y. Fukazawa, H. Kanuka, T. Kato, M. Yoshino, T. Okubo, S. Ogata, H. Kaiya, A. Hazeyama, T. Tanaka, N. Yoshioka, and G. Priyalakshmi, "Systematic literature review of security pattern research," *Information*, vol. 12, p. 36, 01 2021.

[26] M. Riaz, J. King, J. Slankas, and L. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *2014 IEEE 22nd International Requirements Engineering Conference*, vol. 1. Karlskrona, Sweden: IEEE, 2014, pp. 183–192.

[27] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie, "Automated extraction of security policies from natural-language software documents," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: Association for Computing Machinery, 2012.

[28] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, "Polisis: Automated analysis and presentation of privacy policies using deep learning," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 531–548.

[29] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "Policylint: Investigating internal privacy policy

contradictions on google play," in *28th USENIX Security Symposium*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 585–602.

[30] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A survey of deep learning methods for cyber security," *Information*, vol. 10, no. 4, p. 8, 2019.

[31] M. R. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A literature review on mining cyberthreat intelligence from unstructured texts," in *2020 International Conference on Data Mining Workshops (ICDMW)*, vol. 1. Sorrento, Italy: ACM, 2020, pp. 516–525.

[32] C. Suh-Lee, J.-Y. Jo, and Y. Kim, "Text mining for security threat detection discovering hidden information in unstructured log messages," *2016 IEEE Conference on Communications and Network Security*, vol. 2, pp. 252–260, 2016.

[33] S. Dong, P. Wang, and K. Abbas, "A survey on deep learning and its applications," *Computer Science Review*, vol. 40, pp. 300–379, 2021.

[34] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Survey*, vol. 51, no. 5, sep 2018.

[35] L. J. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, 2018.

[36] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: A comprehensive review," *ACM Computing Survey*, vol. 54, no. 3, Apr. 2021.

[37] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *International Conference on Learning Representations*. Vancouver, Canada: OpenReview.net, 2018.

[38] 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*, pages 1536–1547.

[39] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proc. 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Jun. 2021, pp. 2655–2668.

[40] Y. Wang, W. Wang, S. R. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Cana, Dominican Republic, 2021,

[41] A. Habib and M. Pradel, "Neural bug finding: A study of opportunities and challenges," *CoRR*, vol. abs/1906.00307, 2019.

[42] J. K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, "CORE: automating review recommendation for code changes," in *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020 , 284-295.

[43] T. H. M. Le, H. Chen, and M. A. Babar, "Deep learning for source code modeling and generation: Models, applications, and challenges," *ACM Computing Survey*, vol. 53, no. 3, Jun. 2020.

[44] A. T. Nguyen, P. C. Rigby, T. Nguyen, D. Palani, M. Karanfil, and T. N. Nguyen, "Statistical translation of english texts to api code templates," in *2018 IEEE International Conference on Software Maintenance and Evolution*, 2018, pp. 194–205.

[45] C. Bhadane, H. Dalal, and H. Doshi, "Sentiment analysis: Measuring opinions," *Procedia Computer Science*, vol. 45, pp. 808–814, 2015.

[46] S. M. Rezaeinia, R. Rahmani, A. Ghodsi, and H. Veisi, "Sentiment analysis based on improved pre-trained word embeddings," *Expert Systems with Applications*, vol. 117, pp. 139–147, 2019.

[47] F. Luo, P. Li, P. Yang, J. Zhou, Y. Tan, B. Chang, Z. Sui, and X. Sun, "Towards fine-grained text sentiment transfer," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2013–2022.

[48] J. Li, R. Jia, H. He, and P. Liang, "Delete, retrieve, generate: A simple approach to sentiment and style transfer," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1865–1874, 2019. New Orleans, Louisiana. Association for Computational Linguistics.

[49] S. Negi, M. de Rijke, and P. Buitelaar, "Open domain suggestion mining: Problem definition and datasets," *CoRR*, vol. abs/1806.02179, 2018.

[50] G. Zingle, B. Radhakrishnan, Y. Xiao, E. F. Gehringer, Z. Xiao, F. Pramudianto, G. Khurana, and A. Arnav, "Detecting suggestions in peer assessments," in *Proceedings of the the 12th International Conference on Educational Data Mining*, 2019.

[51] B. Lin, F. Zampetti, G. Bavota, M. Di Penta, and M. Lanza, "Pattern-based mining of opinions in q&a websites," in *2019 IEEE/ACM 41st International Conference on Software Engineering*, 2019, pp. 548–559.

[52] W.-H. Khong, L.-K. Soon, H.-N. Goh, and S.-C. Haw, "Leveraging part-of-speech tagging for sentiment analysis in short texts and regular texts," in *Semantic Technology*, R. Ichise, F. Lecue, T. Kawamura, D. Zhao, S. Muggleton, and K. Kozaki, Eds. Cham: Springer International Publishing, 2018, pp. 182–197.

[53] C. Nicholls and F. Song, "Improving sentiment analysis with part-of-speech weighting," in *2009 International Conference on Machine Learning and Cybernetics*, vol. 3, 2009, pp. 1592–1597.

[54] N. Silva, D. Ribeiro, and L. Ferreira, "Information extraction from unstructured recipe data," in *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*. NY, USA: Association for Computing Machinery, 2019, p. 165–168.

[55] K. Kalaivani, S. Uma, and C. Kanimozhiselvi, "A review on feature extraction techniques for sentiment classification," in *2020 Fourth International Conference on Computing Methodologies and Communication*, 2020, pp. 679–683.

[56] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Survey*, vol. 53, no. 3, jun 2020.

[57] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[58] H. Edwards and A. J. Storkey, "Towards a neural statistician," in *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net, 2017.

[59] B. Hariharan and R. B. Girshick, "Low-shot visual object recognition," *CoRR*, vol. abs/1606.02819, 2016.

[60] Z. Ji, X. Zou, T. Huang, and S. Wu, "Unsupervised few-shot learning via self-supervised training," *CoRR*, vol. abs/1912.12178, 2019.

[61] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 3637–3645.

[62] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 4080–4090.

[63] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

[64] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Prefixspan,: mining sequential patterns efficiently by prefix-projected pattern growth," in *Proceedings 17th International Conference on Data Engineering*, 2001, pp. 215–224.

[65] L. Richardson, *Beautiful Soup Documentation*, 2022, https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[66] Oracle, *Java SE Development Kit 11 Documentation*, 2021, https://www.oracle.com/java/technologies/javase-jdk11-doc-downloads.html.

[67] Python Software Foundation, *Python 3.7.12 Documentation*, 2021, https://docs.python.org/3.7/.

[68] P. Garg and S. H. Sengamedu, *Resource leak detection in Amazon CodeGuru Reviewer*, 2021, https://aws.amazon.com/blogs/devops/resource-leak-detection-in-amazon-codeguru/.