Topical: Learning Repository Embeddings from Source Code using Attention

Agathe Lherondelle*, Yash Satsangi*, Fran Silavong*, Shaltiel Eloul*, Sean Moran†

JPMorgan Chase, London, UK

Email: *firstname.lastname@jpmchase.com, †sean.j.moran@jpmchase.com

Abstract—Machine learning on source code (MLOnCode) promises to transform how software is delivered. By mining the context and relationship between software artefacts, MLOnCode augments the software developer's capabilities with code auto-generation, code recommendation, code auto-tagging and other data-driven enhancements. For many of these tasks a script level representation of code is sufficient, however, in many cases a repository level representation that takes into account various dependencies and repository structure is imperative, for example, auto-tagging repositories with topics or auto-documentation of repository code *etc*. Existing methods for computing repository level representations suffer from (a) reliance on natural language documentation of code (for example, README files) (b) naive aggregation of method/script-level representation, for example, by concatenation or averaging. This paper introduces *Topical* a deep neural network to generate repository level embeddings of publicly available GitHub code repositories directly from source code. Topical incorporates an attention mechanism that projects the source code, the full dependency graph and the script level textual information into a dense repository-level representation. To compute the repository-level representations, Topical is trained to predict the topics associated with a repository, on a dataset of publicly available GitHub repositories that were crawled along with their ground truth topic tags. Our experiments show that the embeddings computed by Topical are able to outperform multiple baselines, including baselines that naively combine the method-level representations through averaging or concatenation at the task of repository auto-tagging. Furthermore, we show that Topical's attention mechanism outperforms naive aggregation methods when computing repository-level representations from script-level representation generated by existing methods. Topical is a lightweight framework for computing repository-level representation of code repositories that scales efficiently with the number of topics and dataset size.

I. Introduction

Code hosting websites such as GitHub, have revolutionized code development, providing developers with a collaborative environment within which they discover relevant code, follow technological advances, learn, and incubate ideas for new applications. The amount of open source repositories is massive, for example GitHub hosts over 200 million repositories[1]. Automatic tools, that enable faster and efficient search-based access to relevant repositories, such as auto-tagging repositories [1] with semantic keywords, modelling their topics [2] are crucial for managing this deluge of information. The challenge of processing source code for useful applications is addressed by the field of MLOnCode [3] leading to a variety

of useful applications such as duplication detection [4], design patterns for software development [5], code quality and auto-generation [6], and the extraction of software developer skill sets directly from their code [3]. One of the main challenges for development of these tools and applications is to establish a suitable way for representing code, such as code embeddings or repository embeddings [7], [8].

The main focus of MLOnCode research in recent years has been on predictive tasks at *method-level* or *snippet-level* [9]–[11] granularity. Recently, deep neural network [12] have shown an impressive ability to understand and manipulate method-level code snippets [3], [9], [10]. The progress in code embedding can be linked to the introduction of transformers with *attention mechanisms* in natural language models, such as BERT [13] and related models [14]–[16]. Similar neural models are adapted also for source code embedding [9], [10], [17]–[21]. These models achieve impressive results at a script or a method level, however, only a small number of these works [1], [7] address the task of aggregating information from many scripts into a repository level representation. The methods [1], [7], [22] that address the challenge of computing repository level representation, largely rely on the natural language documentation of the code, for example, the READMEs and other documentation that accompany a typical code repository.

Purely textual information as might be found in README files or logs, does hold important context on the code, however text is used in these models as a 'proxy' information for the code itself. Relying on READMEs puts the onus for correct and efficient documentation on the developer, arguably adding to their work. Code documentation can be tedious and it is not a surprise that many GitHub repositories either completely lack a README or have inaccurate or inconsistent READMEs. In fact, it is not uncommon for repositories to have no documentation at all, or even contain 'inaccurate' documentation (for example, using non-relevant tags to increase visibility) that can easily mislead methods that rely heavily on such information.

How can we effectively generate repository level representations directly from source code? This paper attempts to answer this research question. We introduce a new tool that we call *Topical*, that learns a flexible repository representation which can be applied to multiple downstream tasks such as repository tagging and summarising source code. Topical learns a repository-level embedding by leveraging three domains

---

[1] https://en.wikipedia.org/wiki/GitHub, www.GitHub.com

in a code repository: 1) the encoding of the *Dependencies* as a graph of functions calls within and across scripts or libraries, 2) the *logical code content and structure*, and 3) associated *docstrings and method or file names* in the code. Topical generates an embedding of these three domains for each script file and by using an attention mechanism it yields a repository level representation. In order to train Topical we identify the task of auto-tagging publicly available Github repositories since the repositories and their curated tags are publicly available on Github. We crawl a dataset of these Github repositories along with their ground-truth tags. Our experiments show that Topical with its attention mechanism significantly outperforms baselines that average or concatenate method-level representations for auto-tagging code repository. Topical does not rely on developers to generate READMEs with a natural language documentation of their code and thus eases the burden of code documentation for software developer.

In summary:

- We propose Topical: an attention-based deep neural network architecture that extracts and combines information from three domains in a code repository: 1) dependencies; (2) code content; (3) docstring to generate repository level representations. Topical comes together with a GitHub crawler that extracts repositories along with curated 'featured-topics' that are used to train Topical to generate repository level embedding.

- We show that Topical can be used as a general-purpose lightweight framework for computing repository level representation by combining any other existing method-level representation, for example, import2vec [8]. Our experiments show that taking an average/sum or concatenation of method-level representation to compute repository level embedding can suffer significantly when compared to the attention mechanism employed by Topical.

- We show that Topical outperforms multiple baselines for a GitHub repositories auto-tagging task. Not just Topical outperform naively aggregated repository-level representations from existing embeddings, but Topical also outperforms repository-level representations generated by using attention mechanism with existing method-level representations.

The remainder of the paper is organised as follows: in Section II, we introduce the crawler and our dataset and we discuss the Topical model architecture. Later, in our experimental results (Section III), we benchmark Topical on the database and compare it to TF3D, a novel baseline model that we suggest is a strong basis for comparison to Topical. In Section IV we provide conclusions and pointers for further research in this field.
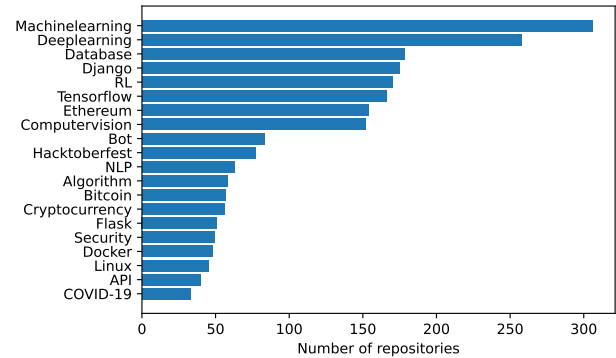
II. METHODOLOGY

A. Dataset and GitHub Crawler

We start by generating a repository level database along with the annotated topics for the repository. Existing open
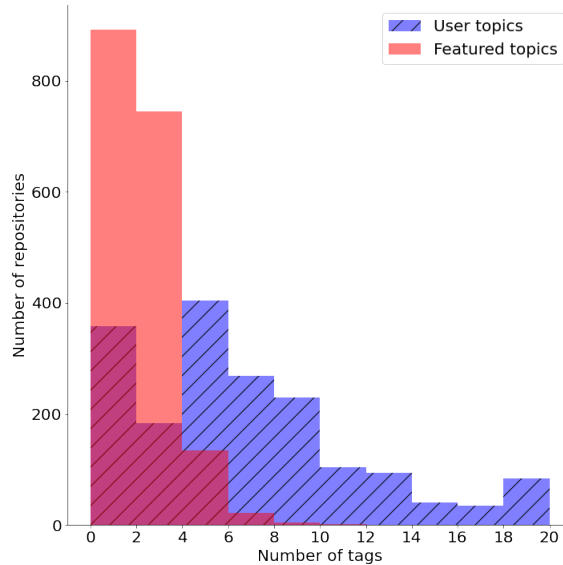
(a)

(b)

Fig. 1: (a) Distribution of the number of occurrences for the top 20 featured topics (red), user topics (blue), before and after Fuzzy Matching (mixed). (b) Distribution of repositories by topics in the dataset.

source code datasets [11], [21] are designated to tasks at a file or method level and do not include sufficient information for studying a collection of files, such as imports, or metadata and commit/git history. At this stage, we only consider Python repositories, however our method can be easily extended to other languages. To our knowledge, there is no benchmark database which includes repositories and their annotation that can be directly applied to this study. To facilitate our study and future research on topic modelling of source code, we build a GitHub crawler tool for generating the database from open-source repositories along with the database composed for this study.

GitHub repositories are often classified by its owner using user-defined topics, which can contain abbreviations, typos, and repetitions. Because of the large variations in topic names, GitHub also defines 480 featured topics, a limited number of predefined topics to be associated with the repository by its
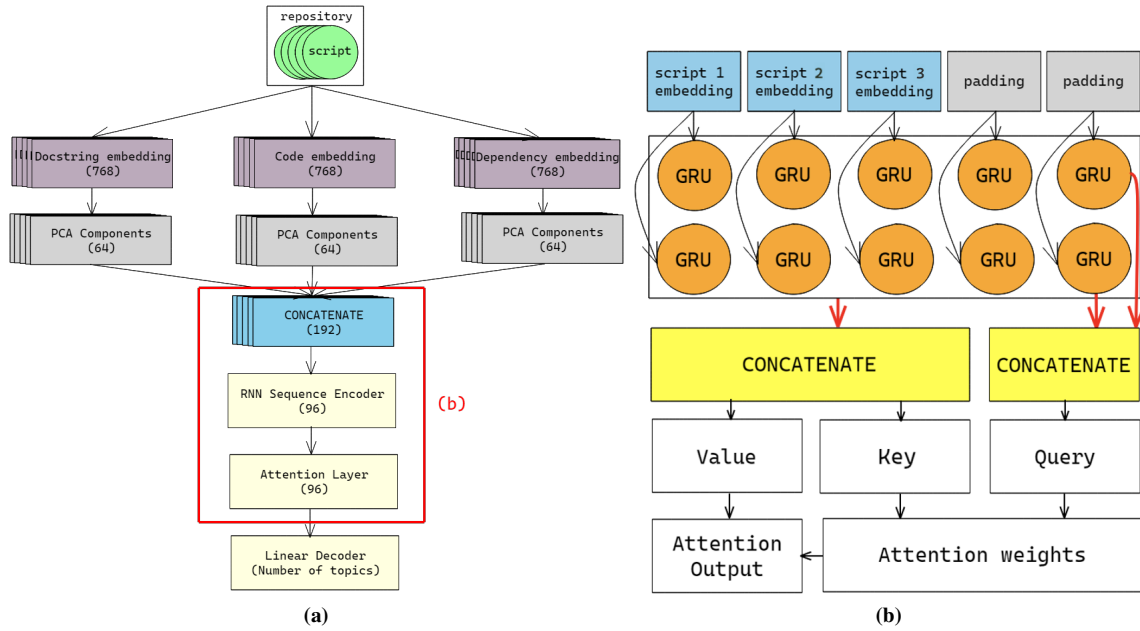
Fig. 2: (a) General model diagram including downstream classification layer. (b) RNN sequence encoder on script embeddings with Attention layer to compute a single embedding representation of the repository.

owner. In order to have a standard label set, the crawler maps the user-defined topics by user to the GitHub featured topics using string matching method [23], [24] relying on threshold (90%) of "Fuzzy matching" (Levenshtein ratio [25]) distance of words to identify similar topics. Figure 1a compares the distributions of the number of topics associated to a repository before and after the mapping to featured topics. The decrease in the number of repositories with a given number of tags in Fig. 1a shows that GitHub users have a tendency to associate numerous topics, which can actually be very similar, to their repository in order to increase their visibility in the GitHub search engine. Note that most of the user-defined topics are effectively related to a featured topic. Furthermore, a significant amount (approximately 32%) of repositories have been found to not be associated with any of the topics and this number increases when restricting to featured-topics. Topical can thus assist the user to automate the process of generating topic tags amongst a limited set. To collect the dataset, we start with an initial set of 20 topics (for example, ML, NLP, Database *etc.*) and for each topic the crawler collects a fixed number of repositories associated with it. The resulting dataset consists of approximately 3000 repositories with approximately 92383 total python scripts across all repositories. However, around 32% of these repositories did not have a featured topic. Since a crawled repository can have multiple featured topics associated with it the resulting distribution of featured topics is skewed towards certain topics. The top 20 most frequent featured topics were selected for the final classification task and the dataset corresponding to these topics consisted of 1600 repositories. At the end, the topic distribution is shown in Fig.1(b). The distribution shows that the last few topics are quite rare as compared to the first few topics but they still

constitute sufficient repositories so that for a classifier to have a overall good score it must classify all the topics accurately.

B. Model

Our main objective is to be able to represent source code at a repository level. Our model for this, Topical, is thus composed of encoders and a deep neural network-based attention mechanism. Figure 2 presents the architecture diagram of the various components of the model. The model can be divided into three stages. In the first stage, we embed the scripts contained in a repository. For that, we leverage pre-trained BERT base transformer models to generate embedding vectors for each scripts in a repository, specifically three embedding vectors are generated for each script as described later (section II-C). In the second stage, we introduce an attention mechanism to obtain a collective embedding from the embedding computed on each of its scripts. In the last stage, we add a classifier unit for the task of multi-label/single-label classification of the repository topics. We detail each of these main stages below.

C. Script-level Embedding

Topical utilises three domains in a repository as inputs to the encoder: source code content and structure, the textual information in the source code, and the dependency graph between scripts by using methods calls to methods in the script, to the class, or to an external library. The code content and textual information (*e.g.* docstrings, filenames) are typically used in code classification [10]. The dependency graph domain captures much of the repository structure and pattern that is useful not only for topic tagging, but also for recognition of patterns in software architectures [8]. Thus we assume that using all three domains will provide a multi-purpose and

comprehensive embedding of a repository. Figure 3 illustrates the way the different domains of the code contained in a script (Fig. 3a) are separated and processed (Fig. 3b) and then tokenized using the appropriate representation of the information and tokenizer (Fig. 3c).

1) Code content and structure: The code content is embedded using the GraphCodeBERT RoBERTa base [10]. GraphCodeBERT base is a BERT model pre-trained on multiple code languages and its tokenizer combines both the raw code content of methods and the 'dataflow' information. The 'dataflow' provides a shallow relational graph of variables within a code method. In this case, we use 512 input tokens size for each script as this is the default for using pre-trained GraphCodeBERT.

2) Textual information - Docstrings Embedding: Although GraphCodeBERT also processes comments in code, here we designate a separate embedding for retrieving textual information from comments and pre-processed method names. This allows us to pre-process textual information found in source code to target repository-level tasks. We extract docstrings, function textual names and file names. In order to obtain a fixed-size final embedding, we integrate file names and method names into a single sentence and separate them from the script docstrings using the special token for separation and encode the tokenized input in DistilBERT [26] (a natural language embedding, which is a pre-trained model on English language). Similar to code embedding, we concatenate comments and function names from a script and use maximum of 512 tokens size as vector input for DistilBERT.

3) Dependency graph embedding: Previous work has explored library import statements to obtain the embedding for imported libraries as dependencies [8] by listing the loaded packages as abstract trees at a repository level. However, it relies only on the package loading statements in a repository. This can substantially misrepresent the actual code communication and library usage in the code. Here we introduce an embedding of the full communication graph between methods and scripts in a software repository. For each script, we retrieve its corresponding edges that link all methods in a script to other classes and other methods, implemented in the same repository but also in external library calls as presented in Fig. 3a-b. To obtain such a graph, we utilize PyCG which is an open-source library to extract dependency graph from static python codes [27]. Because of the descriptive nature of methods and package names, we use the DistilBERT model pre-trained on English language to embed the graph.

In order to tokenize a dependency graph into the DistilBERT model, we dedicate a special token to indicate the link between two nodes in the graph, a method name and its class imports or other method usage. All first rank edges of the graph are then concatenated sequentially with the separating token, before being passed to the DistilBERT model. Figure 3b presents how we retrieve the nodes from the PyCG output and Fig. 3c presents how we tokenize them after introducing $[C]$ as a DistilBERT special token. The usage of the path as a sentence in a trained natural language model is assumed to be desirable

```
from collections import Counter
import time
import logging


def all_unique(nums):
    """returns True if all elements of a list
    are unique otherwise, return False"""
    begin = time.time()
    count = dict(Counter(nums))
    for i in count.values():
        if i > 1:
            logging.info(time.time() - begin)
            return False
    logging.info(time.time() - begin)
    return True
```
(a)

```
{
    "test_pycg": [],
    "test_pycg.all_unique": [
        "collections.Counter",
        "<builtin>.dict",
        "time.time",
        "logging.info"
        ],
    "time.time": [],
    "collections.Counter": [],
    "<builtin>.dict": [],
    "logging.info": []
}
```
(b)

```
['test','_','p','##y','##c','##g','all','_','unique','[C]
','collections','counter','time','time','logging','info']
['returns','true','if','all','elements','of','a',
'list','are','unique','otherwise',',',',','return','false']
['from','Gcollections','Gimport','GCounter','C','import',
'Gtime','C','import','Glogging','CC','C','def','Gall',
'_','unique','(','n','ums', ...]]
```
(c)

Fig. 3: (a) Code snippet. (b) PyCG output dependencies graph on the code snippet. (c) Tokenized sequences for PyCG dependencies, docstring and source code.

as we expect to obtain embedding with "some" relation to the distance between function calls (also words).

D. Repository-level Embedding

Topical applies an attention mechanism to produce a hybrid embedding from different pre-trained (GraphCodeBERT, DistilBERT) BERT models embeddings as discussed above for each script in a repository. The detailed scheme of the model is presented in Fig. 2a-b. We reduce the number of components from each embedding to contribute to our final script representation. This is mainly to reduce computation and make our model scale to large datasets efficiently. Reducing the number of components in the embedding reduces the number parameters and can be optimised to the specific downstream task. We use PCA (Principal Component Analysis) to reduce the dimensionality of 768 embedding vectors to 192 dimensions. After the dimensionality reduction, we combine the information from various scripts into an single dense repository representation using an attention unit. The attention unit includes an RNNs sequence encoder followed by a self-attention mechanism [28]. We use bi-directional Gated Recurrent Units (GRU) [29] as the recurrent unit since a previous study [30] prefer GRU units to achieve better performance

on small datasets of large sequences. GRU units permit the embedding of many scripts as a sequence representing the repository. The self-attention layer allows Topical to optimize the distribution of weights for the scripts. This is especially useful for classifying topics as the 'topic' can manifest in a small fraction of the repository or by distinctive relationship between topics in a collection of scripts. Our attention paradigm is detailed in Fig. 2b. For a given repository, R, we obtain $R^d = \{x_0, x_1, \ldots, x_n\}$, as the script embedding for each domain d, where d belongs to either code structure, docstrings, or dependencies. x_t is therefore a single obtained script embedding, where t is the script position in a sequence of scripts from a repository which is used as an input to the GRU hidden layer. Since we do not consider here the order of the scripts sequence, we utilise attention on the bi-directional GRUs. The hidden layer of the forward GRU is represented as $\overrightarrow{h_t}$, and the hidden layer of the backward GRU is represented as $\overleftarrow{h_t}$, where we calculate:

$$\overrightarrow{h_t} = GRU(x_t, \overrightarrow{h_{t-1}}) \tag{1}$$

$$\overleftarrow{h_t} = GRU(x_t, \overleftarrow{h_{t-1}}) \tag{2}$$

For a repository composed of n scripts, we retrieve the last hidden state of both hidden layers and concatenate them as follows:

$$h_n = [\overrightarrow{h_n}, \overleftarrow{h_n}] \tag{3}$$

We also retrieve the output of the GRU which is the tensor of all its hidden states:

$$y = [h_i]_{0 \leq i \leq n}, h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}] \tag{4}$$

Indeed, h_n contains information from all the other hidden states and thus permits to represent the entire collection of scripts. y is then used as the key and value to the attention layer while the last hidden state h_n is used as query.

1) Sampling scripts from repository: As y should have a fixed-shape, in the case where the total number of scripts is smaller than n, we add padding embeddings (*e.g.* script embedding full of zeros) into the sequence. In this work we empirically choose $n = 15$ as the maximum number of script files to be used for embedding a single repository. Repositories may contain many more scripts, but using low number allows to include raw repositories that are work-in-progress. If we choose to randomly sample scripts from large repositories there is a chance that files from a copied third party libraries would dominate the sampling. To minimise this effect we utilise PyCG once again to sample scripts for the embedding process. This is achieved using paths from the top directory in the repository and choosing scripts that are involved in the function calls path. Scripts are retrieved in the order they are presented in the path. After exploring the path we sample a new path randomly until we populate all 15 scripts.

We apply an attention mask onto these padding embeddings when computing the attention output [28]. The mask matrix

computation sets attention weights to 0 on padding embeddings using attention as:

$$F = softmax(\frac{Q \times k^T + M}{\sqrt{d_k}}) \times V \tag{5}$$

where M the mask matrix is:

$$M_{t,i} = \begin{cases} 0 & \text{if } x_t \text{ is a script embedding} \\ -\infty & \text{if } x_t \text{ is a padding embedding} \end{cases} \tag{6}$$

and Q is the query matrix, k the key vector, V the value matrix as shown in Fig. 2(b) and d_k is the dimension of the key vector.

2) Multi-label Classification: Most of the repositories available on GitHub belong to more than one topic. Furthermore, some featured-topics are subtopics of others (Fig. 1b). For example, NLP focused repository will likely be assigned to the broader topic machine learning. We picked 5-20 representative topics by their frequency that can also be found separate to other topics. The topics chosen for various tests are summed later in Table I. To enable the multi-label classification task, we add a linear layer on top of the attention mechanism, paired with a sigmoid activation function. The whole architecture is trained to minimize the cross entropy loss between the predictions and the ground truth label. Using a validation set, we fix a threshold on the sigmoid output of the decoder. This threshold is optimized by maximizing the F1-score on a validation set between ground-truth binary label vector l and output vector of scores s, converted into binary label vectors $\hat{l}$. For each topic i and output score s_i, the predicted topic label l_i is computed as follows:

$$\hat{l}_i = \begin{cases} 1 & \text{if } s_i \geq threshold \\ 0 & \text{else} \end{cases} \tag{7}$$

E. Baselines

In order to provide conclusive results for our attention based model, we develop multiple competitive baseline models, TF3D, GraphCodeBert, and Import2vec. TF3D is a model based on term frequency. This model allows us to compare Topical to a non deep learning model, but with similar embedding information (source code, docstrings, and dependencies). The second baseline, GraphCodeBERT uses embedding based only on code content of the repository. Finally, we use the repository embedding model, Import2vec, in four new variations which are built on top of the pre-trained Import2vec model. The performance of these baselines highlights the flexibility and effectiveness of Topical when combined with existing script/file level embedding.

1) TF3D - A Statistical NLP Baseline: In order to compare the deep BERT embedding with attention mechanism to a traditional statistical model, we develop a competitive statistical baseline called TF3D. TF3D is based on representing terms frequencies, such as TF-IDF, but adapted in this case to collection of scripts/methods. Typical statistical term-based models for source code were recently shown to be effective for source code analysis and similarity detection [31]–[33].

Similar to the attention model we combine three source-code feature domains: (a) the code structure, by using AST (Abstract Syntax tree) features of each method in code; (b) The *docstrings*, which are any comments at the method level, and function names and finally, (c) dependency/libraries of script files. We represent a repository as a collection of n methods with their corresponding feature vectors (m) in the source code of a repository, i, $R_i^d = \{m_1, m_2, \ldots, m_n\}$ where d belongs to either code structure, docstring or dependencies. For each repository in each d, R_i^d, we use aggregation to represent the probability vector of features in a repository:

$$S_i^d = \frac{\sum_j m_j}{\|m\|} \tag{8}$$

By sampling a training set of N repositories $\{R_0, R_1, \ldots, R_N\}$ we calculate the terms matrix, $C(3 \times n_T)$, for each topic (T) in n_T topics by using the arithmetic mean on the logarithm of S_i^d:

$$C(d, T) = \frac{\langle \ln S_{i \in T} \rangle}{\langle \ln S_{i \notin T} \rangle} \tag{9}$$

Note that here we modified the standard TF-IDF [34] and introduce the logarithm to penalise excessively repeating terms in scripts or methods in R_i, that can dominate the frequency vectors (instead of penalising the inverse of the frequency as typically used). Equation 9 is reminiscent of the clarity score from the field of Information Retrieval [35] that measures how different a token distribution is from the background. Then, we calculate the cosine similarity, to obtain the embedding matrix representing each repository, i, in the training and testing sets: $E_i(3 \times n_T) = \frac{S_i \cdot C^T}{\|C\|\|S_i\|}$. Finally, we use the embedding matrices for classification using a standard random forest regressor classifier for multi-label tagging of repositories.

2) GraphCodeBERT: GraphCodeBERT by itself combines code-content, its *data-flow*, and comments found within methods. GraphCodeBERT embeds method level source code for various code-related tasks. We therefore use it as a baseline to show the efficiency of our attention model.

It is noted, that we have experimented with other models for method level source code embedding, such as Code-BERT infused with code AST using Attention mask. In that model, CodeBERT relies only on source code content and using AST graph connections as attention masks enables to capture code data flow. However, the GraphCodeBERT provided significantly better results, and we therefore report only GraphCodeBERT as a competitive baseline that uses pre-trained method level embedding to represent a repository. Compared to Topical that combines the docstring, code and Dependency embedding, GraphCodeBERT baseline only uses the code embedding.

3) Import2Vec: The last baseline we compare to is a model based on Import2vec embedding [8]. These set of baselines also show how Topical architecture can be used with existing method-based embedding to generate repository-level embedding. Import2vec provides vector embedding for software libraries, imported by a script and is based on the

semantic similarity between these libraries. The idea behind learning this embedding is to capture similar software libraries based on how often they are imported alongside each other. These representations are obtained by training a deep neural network to predict the probability that a pair of software libraries are imported together or not in at least one source file in a large dataset of code repositories. Theeten et. al. [8] provide pre-trained Import2vec embeddings that return a vector representation for most existing python libraries/packages, for example, numpy, tensorflow, etc.

In order to use the Import2vec embedding for downstream tasks such as classification in our case, we first extract the list of all the software libraries imported in a repository. For each library imported by the repository we obtain its Import2vec vectors. The dimensionality of these vectors can vary between 60 to 200. [8] reports that embedding size of 100 is sufficient, so we set vectors dimension to be 100 in this work. A straightforward way to aggregate these vectors for classification is either by taking their average/mean or by concatenating them in a fixed order, and then using a classifier. We combine these vectors using the attention mechanism as proposed in the Topical architecture. This can be achieved simply by representing each repository $R = \{x_0, x_1, \ldots, x_n\}$ as a list of Import2vec vectors of the libraries imported by the repository. This repository R now can be processed in same way as described in section 2.4. We obtain 4 variations of the Import2vec embedding as baselines:

- **I2V-conc-linear:** Concatenation of the Import2vec embeddings of all libraries imported by a repository in a pre-defined order and then train a linear neural network layer for auto-tagging of repositories.
- **I2V-conc-attn:** Concatenation of the Import2vec embedding in same way as i2v-conc-linear, however, these embedding vectors are then combined using the Topical attention mechanism architecture as shown in Fig. 2(b) and described in section 2.4.
- **I2V-mean-linear:** The average of the Import2vec embedding vectors of all the repositories imported by a repository followed with a linear layer
- **I2V-mean-attn:** The average of the Import2vec embedding vectors of all libraries imported by a repository and then processed by the Topical attention mechanism.

4) Evaluation Metrics: In the evaluation, we compare the F1-scores based on threshold for overall performance of classification and also evaluate the LRAP (Label Ranking Average Precision) score which computes ranking-based average precision (without the need for a threshold). This is a popular metric used in literature for evaluating multi-label classification [1], [36] The LRAP is calculated as follows:

$$LRAP(y, \hat{f}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \frac{1}{\|y_i\|_0} \sum_{j:y_{ij}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}} \tag{10}$$

$$y \in \{0, 1\}^{n_{samples} \times n_{labels}}$$

$$\hat{f} \in R^{n_{samples} \times n_{labels}}$$

$$\mathcal{L}_{ij} = \{k : y_{ik} = 1, \hat{f}_{ik} \geq \hat{f}_{ij}\}$$

$$\text{rank}_{ij} = |\{k : \hat{f}_{ik} \geq \hat{f}_{ij}\}|$$

where y is the binary indicator matrix of the ground truth labels and $\hat{f}$ is the vector of scores predicted for each label. The rank_{ij} provides the index of the ordered prediction vector, and $|\mathcal{L}_{ij}|$ is the number of true predictions for all indices above rank_{ij}. Thus, the average over the ratio for all samples gives a reliable metric for multi-label scores. LRAP measures the quality of a multi-label classification by first ranking the labels predicted by a model and then reasoning about order in which the correct labels appear. If the correct labels appear at top ranks then a score of 1 (or close to 1) is awarded, however if correct labels appear at lower ranks then, depending on their rank, only a small fraction of 1 is awarded.

F1 scores: We report F1-scores (in addition to LRAP) since it is one of the most commonly reported and well-understood metric for classification tasks. We report the micro-average F1 score [37] computed by counting the global true positive, false positive and false negative across the dataset. In general, LRAP and F1 differ in the way they punish the mistake of a classifier. LRAP punishes these extra assignment of label by considering its rank where as F1 punishes such assignments in 0-1 fashion by computing the global precision.

III. RESULTS

We start by first analysing the embedding output from our model. In the second part, we compare performance of various baselines for auto-tagging on different number of topics. We also study the performance of algorithms that use the attention mechanism in comparison to standard aggregation. The main idea behind these experiments is (a) to examine the performance and effectiveness of the attention mechanism, and (b) the effect of using dependency and docstring embedding in combination with code embedding. Finally, we compare the performance of various algorithms across the size of dataset to compare how well they scale with data. All algorithms are trained by minimising the cross-entropy loss with a fixed learning rate of 0.002 using ADAM optimiser with weight decay [38], [39]

Figure 4 introduces a latent visualization of the repository embedding from Topical, using the attention model. We embed repositories from 5 popular topics (see Table I) crawled directly from GitHub. We use TSNE projection on a 2D latent space, and the top three PCA components for 3D projection. Figure 4 shows a clear separation between embedding projections from Topical in the task for classification of GitHub topics. It is worth noting that two latent components do not fully capture the abstract differences between repositories, such as the context or topic of the repository. However separation is still clearly more pronounced (Figure 4b) with the Topical embedding in comparison to an embedding achieved without the attention mechanism (the inset of Figure 4b).

The PCA projection (Fig. 4a) also approves, at least visually, our decision to reduce dimensionality of the embedding, before the classifier head. The inset of Fig. 4b shows also the
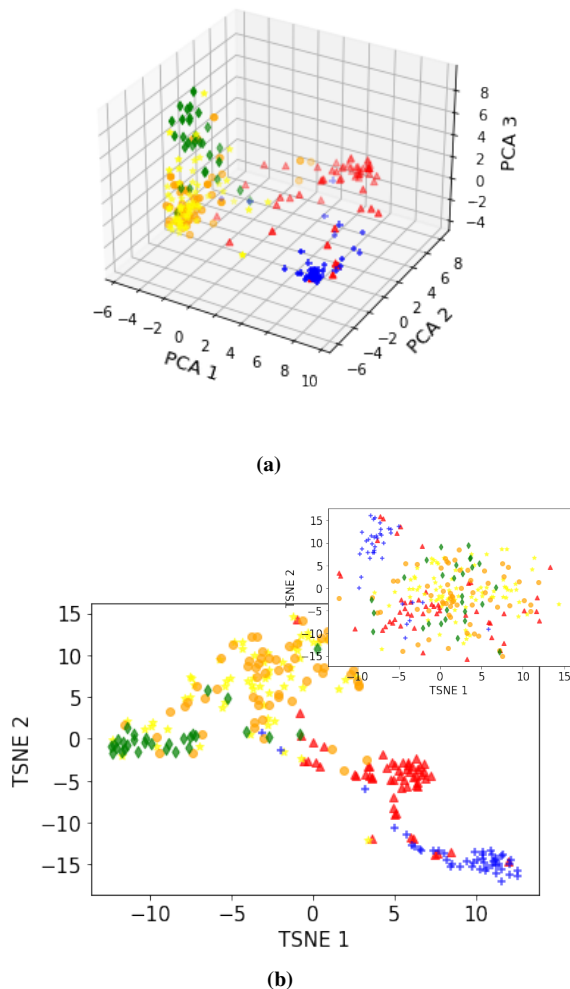

(a)

(b)

Fig. 4: (a) 3D PCA projection of repositories embedding for the top 5 topics by frequency. (b) 2D TSNE projection of repositories embedding for the top 5 topics. Inset shows the 2D TSNE projection for embedding for the top 5 topics without the attention mechanism. The top 5 topics: ML, DL, Database, Django, and RL are represented by Orange, Yellow, Red, Blue and Green points respectively.

projection of the embedding without the attention mechanism. It shows that using the mean embedding (*i.e.* no attention mechanism) can be useful but far less efficient in clustering similar embedding of repositories in comparison to the proposed attention model.

In the first experiment we compare the performance of various baselines for multi-label classification of 5,10,15,20 topics. The topics and their frequency in the dataset are presented in Table I and Fig. 1 respectively. Classification results for Topical with attention model in comparison to the baselines are shown in Figure 5. Topical shows overall better F1-score than TF3D and GraphCodeBERT baselines, and 6%-10% higher performance for LRAP scores. This is also true for F1-scores as Topical does better than the rest of the baselines.

Overall, Import2vec based baselines perform slightly poorer than GraphCodeBERT and TF3D indicating that processing the full source code has significant advantage over only looking at software libraries imported by a repository. A second and important insight that is shown in this experiment is that when combining pre-trained embeddings with the attention mechanism can result in significant performance gain. For example, the performance of Import2vec embedding grows significantly when using the Topical attention as compared to naive aggregation mechanisms such as concatenation and averaging. Another trend revealed by the results of various topics number, is that as the number of topics increases the performance starts to drop. However, one likely reason for that could be the overlap between similar topics.
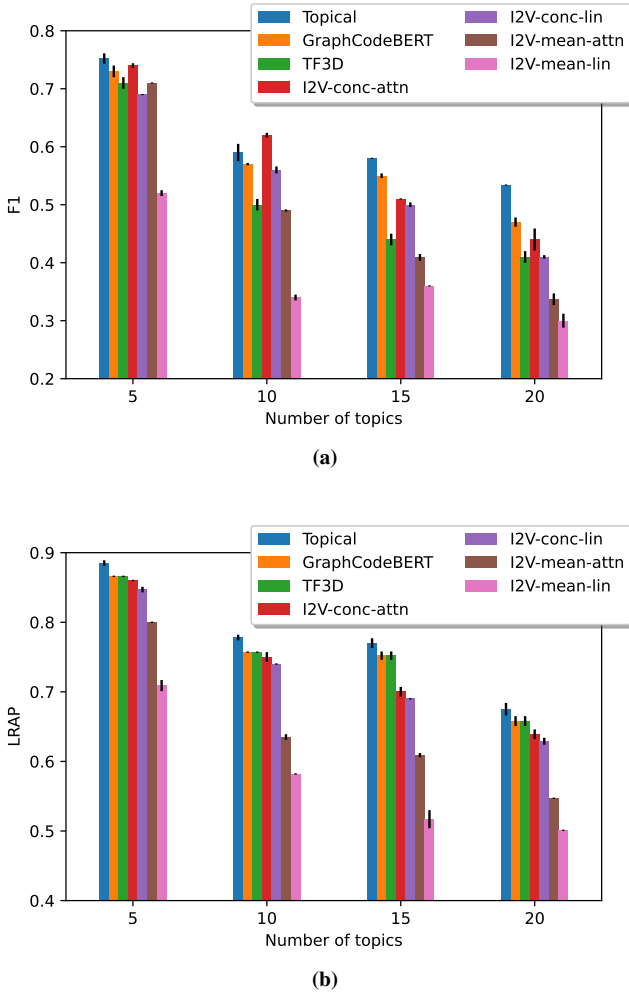
TABLE I: Topics and size of the dataset (number of repository) for the reported experiments. A 70-30 % train-test split is used to generate training and test data. The metrics reported/shown in the figures are computed on the test data.

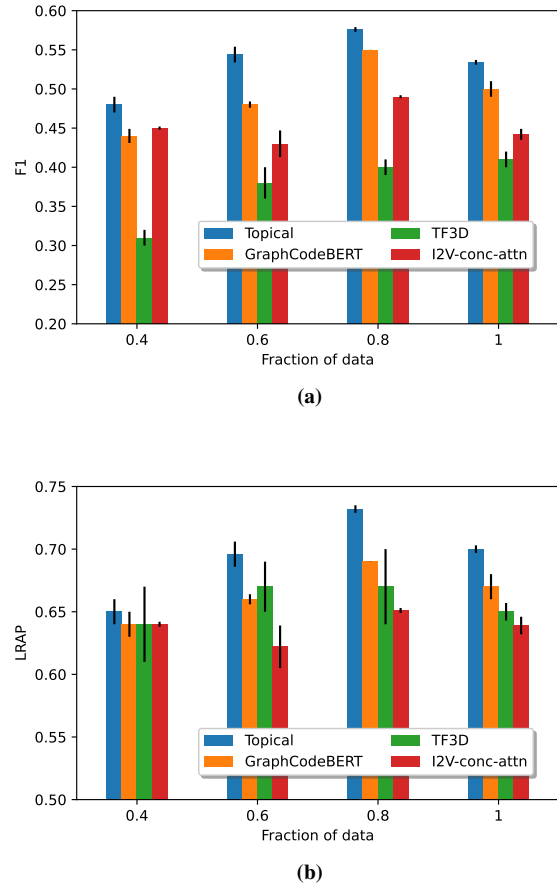| No. of topics	Number of Repositories	Selected topics
5	760	Machine Learning (ML), Deep Learning (DL), Database, Django, Reinforcement Learning (RL)
10	1200	ML, DL, Database, Django, RL, Tensorflow(TF), Ethereum, Computer Vision (CV), Bot, Hacktoberfest
15	1376	ML, DL, Database, Django, RL, TF, Ethereum, CV, Bot, Hacktoberfest, Natural Language Processing (NLP), Algorithm, Bitcoin, Cryptocurrency, Flask
20	1586	ML, DL, Database, Django, RL, TF, Ethereum, CV, Bot, Hacktoberfest, NLP, Algorithm, Bitcoin, Cryptocurrency, Flask, Security, Docker, Linux, API, Covid-19

(a)

(b)

Fig. 5: (a) F1-score comparison and (b) LRAP score comparison for various baselines for auto-tagging for 5,10,15 and 20 topics.

A. Effect of varying data size

We aim to provide a tool that is easily trained on various tasks related to repository embedding. We therefore evaluate our model in comparison to the baselines for various sizes of training set. Figure 6 shows the F1-score and LRAP as

(a)

(b)

Fig. 6: (a) F1-score comparison for 20 topics multi-label classification on 40%, 60%, 80% and full dataset size (b) LRAP score comparison for 20 topics multi-label classification on 40%, 60%, 80% and full dataset size.

function of the fraction of training set for 20 topic classification. We observed that even though we use a deep embedding attention model, it is enough to have a small amount of

repositories to obtain high performance. This means that training and crawling becomes an instant and low cost task, even for training and crawling a dataset for new topics or other classification tasks. TF3D, as expected, is useful in the small data regime, but with increasing of training set, overall, the attention model relying on deep embedding of the repository, provides higher scores. As the size of the dataset increases, the performance of various models also steadily increases, but also slowly saturates. In the future, as we collect more data, we intend to increase the number of parameters used by Topical, in the hope to get even better performance. In a larger dataset further parameter optimisation may be performed to improve performance further such as tuning the dimensionality reduction size and attention size.

B. Discussion

The results sections answers our main research question which can be further broken down into the following questions:

- How can we extract extract useful representation from different sources of information in a code repository? To answer this question we propose Topical that extracts and combines method-level representations and dependencies graph to compute repository level representations. Our results show that Topical outperforms other baselines that includes representation based on only source code information, and libraries imported in a repository at auto-tagging task.
- How can we combine method level representations to generate repository level embedding? Our results show that using Topical can be used to generate repository level representation from existing method level representations (import2vec). Not only Topical is compatible with these existing representation but repository level representation computed by Topical outperform other methods for aggregating method level representations.

Table II describes additional metrics for three attention-based baselines, highlighting a qualitative difference between them. In the table, the precision (averaged over test repositories) [37] is the ratio of number of topics correctly predicted by a model to the total topics predicted by Topical to be associated with a repository. Recall [37] (averaged over test repositories) is the ratio of topics correctly predicted by a model to total topical associated with a repository. Optimized threshold is the threshold used to convert probabilistic output of a model to binary predictions. In general, all three algorithms have good values of recall for a 20 class multi-label classification task, however, it is the precision that decides the overall performance. Even though Topical does not have the highest recall (but quite close to other baselines), it is significantly more precise. Combining these with the LRAP scores suggests that Topical assigns topics with high precision and places them at high ranks. On the other hand, GraphCodeBERT and Import2Vec based baselines seems to pick as many topics as possible but then places them at lower/bottom ranks resulting in a low precision and LRAP score, but high recall. Also,

note that even for 20 topics, Topical (and all attention based baselines) provides an LRAP score greater than 0.6. Loosely speaking, this suggests that these baselines place the relevant topics at top 50% of ranks and thus their poor precision can be possibly improved by not considering the bottom placed topics.

C. Threats to validity

We can identify a few threats to the validity of some of the results in the paper. The difference in performance of the top few algorithms is modest but we re-sample the dataset with repeat experiments to make sure that the standard deviation is sufficiently low and below the difference measured. Changes in performance can also arise as function of the dataset size and number of topics we collected. Our study shows how the performance is affected by a range of data size and number of topics. The results (Figure 5 indicate that the tagging will also be valid in application of large number of topics. However,we note that even when performance is reduced in tagging much larger number of topics, we expect that the main contribution and insights from the paper will hold, that is, a framework for generating repository level embeddings. A threat to validity may also arise from the practical decision to maintain low resources (hardware, computation time). For example, hyper-parameters (padding length) are optimised for experimentation on a portion of the dataset, and used in all experiments. Similarly, our proposed framework is a lightweight framework that does not require training on GPUs/TPUs, and thus, intro-duction of specialised hardware could affect the results due to further optimisation or handling larger databases. Finally, auto-tagging is a multi-label classification and there can be cases when a more specialized method/loss for multi-label classification can affect our observations. Similarly, there can be an argument for a better selection/distribution of topics or even reasoning about sub-topics among topics.

D. Limitations

In the future we would like to scale-up the application of Topical to a larger dataset using GPU based training. Larger and more complex neural network architectures benefit from larger datasets and are typically more adept at tasks with higher difficulty, such as summarisation. Note that even in the absence of these settings, our methodology and insights presented in our study remain valid. Another limitation that we would like to address in the future is testing our methodology for tasks other that auto-tagging, for example, summarisation or finding similar repositories. Finally, we would like to address the relatively low precision compared to recall of some of the methods presented in the paper in the future.

E. Related Work

Previous research for source code topic modelling [40], [41] found success in adapting statistical approaches that have been previously used for the topic modelling of documents for information retrieval (IR) and natural language processing (NLP) tasks [42]–[44]. These techniques treat the source code

TABLE II: Comparison table for Topical model with mean-GraphCodeBERT and TF3D baselines.

Model	Precision	Recall	Optimized threshold
Topical	0.485 (0.017)	0.63 (0.032)	0.217 (0.025)
GraphCodeBERT	0.41 (0.031)	0.67 (0.0)	0.14 (0.01)
I2V-conc-attn	0.35 (0.03)	0.63 (0.03)	0.187 (0.01)

as a collection of independent tokens and learn statistical models of the code using term frequency analysis such as with TF-IDF or Latent Dirichlet Annotation (LDA) [40], [41], [43], [45]–[47]. Ascuncion *et al.* demonstrates the usefulness of LDA for uncovering topics over software artefacts, and how those topics can be useful in the traceability task which seeks to uncover links between related software artefacts links [48]. In a different approach, the automatic tagging of code with topics has been achieved using the 'README' files in GitHub repositories [45]. In order to assess the performance of *Topical* in comparison to the statistical approach (such as LDA, or TF-IDF), we develop a designated term frequency model. We call it here TF3D, as similar to Topical, it leverages terms distributions from three repository domains, the source code, packages imports (dependencies), and docstrings. We detail the model and results later in this work.

Deep neural network based approaches have replaced many term distribution models with large parametric language models, for example, BERT [13], [26]. Pre-trained BERT models have been recently extended by Guo *et al.* to model programming languages. CodeBERT [9] is a multilingual model trained on several leading programming languages. CodeBERT was further improved in the GraphCodeBERT variation, which incorporates code structure information into the model. Graph-CodeBERT [10] achieves state-of-the-art results on tasks related to the CodeSearchNet dataset [21]. Recently, Theeten *et al.* developed the 'Code-Compass' which uses their proposed Import2Vec [8] tool for representing script dependencies in an embedding space, where similar dependencies are clustered together in the space. Each package is associated with a numerical vector which Code-Compass uses to recommend related packages using similarity measurements. Import2Vec [8] is one of the baselines of our study. We show that Import2vec, when combined with Topical's attention mechanism, results in a more effective repository level representation. This is opposed to existing methods for combining script level information, for example, mean or concatenation of script-level information. [7].

Existing solutions [49]–[52] proposed for repository-level tagging typically rely on textual information such as filenames, READMEs and further documentation found in a typical code repository. For example, recent work by Izadi *et al.* [1] tags a repository with topics by harvesting filenames, READMEs and Wiki data. By using DistilBERT on tokenized words, they produced a single embedding for the repository, and add a fully-connected layer completed by a sigmoid activation function to enable multi-label tagging. We differ from these approaches in that Topical only uses script level information (code) without the textual information in README or Wiki data. Extracting useful information from README is an NLP task that puts

the burden of quality on the original README created by the software developer. Furthermore, it makes the learned representation prone to misleading information in the README. Recently, Rokon *et al.* proposed Repo2vec [7] which offers repository level embedding, but it includes documentation such as README files and metadata into the embedding input. In more detail, Repo2Vec combines information from source code, metadata including READMEs and repository structure. It aggregates the three information sources into a single embedding by concatenating them into a single vector. The architecture is then trained on a dataset that consists of human annotated labels of whether two given repositories are similar or not. In this work, we train topical on an automatically generated dataset which we provide. Our crawler already extracts GitHub curated topics associated with repositories and thus does not require human annotations which can be difficult to scale and to evaluate. Most importantly, Topical combines the information from 'content only' *i.e.* the source code, repository structure and associated docstring in source codes. Here, by using an attention-mechanism, Topical yields significantly better classification results than embedding vectors that are aggregation of information via a mean or concatenation as shown by our experiments. It is therefore presumed here that an attention mechanism could potentially outperform repo2vec for similarity tasks. Finally, we focus on repositories that do not necessarily have a sufficient or reliable documentation in READMEs (or any documentation at all), and mainly encode source code and repository structure in a dense embedding which is a highly reliable and robust representation of a repository. Note that we automatically crawl repositories for our database which is not limited to documented repositories only.

IV. CONCLUSIONS

In this paper we introduce Topical, a tool for generating repository level embedding directly from the repository source code. Topical uses curated topics from GitHub that are crawled along with the repositories to obtain repository level embedding that can be used for further downstream tasks. Topical outperforms a competitive statistical model, TF3D, and the Import2Vec baselines that we also introduce in this paper as part of our contributions. Topical's embedding, whilst also bridging different scales of code representation from function to script-level, also operates across different domains of code representation, source code, docstrings, and dependencies. Our Github crawler and curated dataset of repositories are also shared to the community to facilitate further research, as well as the code for our proposed models.

One of our key insights is that the use of attention for combining script level embedding can boost performance for

downstream tasks. Notably, the repository embedding vectors use pre-trained BERT models which result in an easy and transferable package to be used with low cost manner and without special hardware in other downstream tasks, or additional topics. Future research will investigate other downstream tasks based on Topical embedding for repositories, such as summarising a repository with a fluent natural language sentence based on the content of the repository and determining the range of skill-sets exhibited by developers who contributed to the repository.

REFERENCES

[1] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," 2021.

[2] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol. 80, pp. 457–479, 2014.

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3212695

[4] D. Spinellis, Z. Kotti, and A. Mockus, "A dataset for github repository deduplication," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 523–527.

[5] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Guéhéneuc, "Studying software engineering patterns for designing machine learning systems," in *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. IEEE, 2019, pp. 49–495.

[6] T. H. Le, H. Chen, and M. A. Babar, "Deep learning for source code modeling and generation: Models, applications, and challenges," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–38, 2020.

[7] M. O. F. Rokon, P. Yan, R. Islam, and M. Faloutsos, "Repo2vec: A comprehensive embedding approach for determining repository similarity," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 355–365.

[8] B. Theeten, F. Vandeputte, and T. Van Cutsem, "Import2vec: Learning embeddings for software libraries," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 18–28.

[9] 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," 2020.

[10] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," 2021.

[11] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. R. Choudhury, L. Decker, V. Thost, L. Buratti, S. Pujar, and U. Finkler, "Project codenet: A large-scale AI for code dataset for learning a diversity of coding tasks," *CoRR*, vol. abs/2105.12655, 2021. [Online]. Available: https://arxiv.org/abs/2105.12655

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[13] 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://aclanthology.org/N19-1423

[14] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. [Online]. Available: https://aclanthology.org/N18-1202

[15] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[16] 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," 2020.

[17] A. Kanade, P. Maniatis, G. Balakrishnan, and K. Shi, "Learning and evaluating contextual embedding of source code," 2020.

[18] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 1433–1443. [Online]. Available: https://doi.org/10.1145/3368089.3417058

[19] L. Buratti, S. Pujar, M. A. Bornea, J. S. McCarley, Y. Zheng, G. Rossiello, A. Morari, J. Laredo, V. Thost, Y. Zhuang, and G. Domeniconi, "Exploring software naturalness through neural language models," *CoRR*, vol. abs/2006.12641, 2020. [Online]. Available: https://arxiv.org/abs/2006.12641

[20] R. M. Karampatsis and C. Sutton, "Scelmo: Source code embeddings from language models," 2020.

[21] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," 2020.

[22] Y. Zhang, F. F. Xu, S. Li, Y. Meng, X. Wang, Q. Li, and J. Han, "Higitclass: Keyword-driven hierarchical classification of github repositories," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 876–885.

[23] A. Cohen, "Fuzzywuzzy: Fuzzy string matching in python, july 2011," *URL http://chairnerd. seatgeek. com/fuzzywuzzy-fuzzy-string-matching-in-python/*, vol. 1, 2014.

[24] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, no. 1, p. 31–88, mar 2001. [Online]. Available: https://doi.org/10.1145/375360.375365

[25] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[26] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.

[27] V. Salis, T. Sotiropoulos, P. Louridas, D. Spinellis, and D. Mitropoulos, "Pycg: Practical call graph generation in python," 2021.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. 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.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[30] S. Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example," in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, 2020, pp. 98–101.

[31] D. Azcona, P. Arora, I.-H. Hsiao, and A. Smeaton, "user2code2vec: Embeddings for profiling students based on distributional representations of source code," in *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, 2019, pp. 86–95.

[32] D. Fu, Y. Xu, H. Yu, and B. Yang, "Wastk: a weighted abstract syntax tree kernel method for source code plagiarism detection," *Scientific Programming*, vol. 2017, 2017.

[33] M. M. Islam and R. Iqbal, "Socer: A new source code recommendation technique for code reuse," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 1552–1557.

[34] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

[35] S. Cronen-Townsend, Y. Zhou, and W. B. Croft, "Predicting query performance," in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 299–306. [Online]. Available: https://doi.org/10.1145/564376.564429

[36] Z. Chu, K. Stratos, and K. Gimpel, "Natcat: Weakly supervised text classification with naturally annotated resources," *arXiv preprint arXiv:2009.14335*, 2020.

[37] H. Daumé, *A course in machine learning*. Hal Daumé III, 2017.

[38] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[40] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 522–531.

[41] T.-H. Chen, S. W. Thomas, and A. E. Hassan, "A survey on the use of topic models when mining software repositories," *Empirical Software Engineering*, vol. 21, no. 5, pp. 1843–1919, 2016.

[42] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[43] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, no. null, p. 993–1022, Mar. 2003.

[44] X. Yi and J. Allan, "A comparative study of utilizing topic models for information retrieval," in *European conference on information retrieval*. Springer, 2009, pp. 29–41.

[45] A. Sharma, F. Thung, P. S. Kochhar, A. Sulistya, and D. Lo, "Cataloging github repositories," ser. EASE'17. New York, NY, USA: Association for Computing Machinery, 2017, p. 314–319. [Online]. Available: https://doi.org/10.1145/3084226.3084287

[46] A. Hindle, M. W. Godfrey, and R. C. Holt, "What's hot and what's not: Windowed developer topic analysis," in *2009 IEEE international conference on software maintenance*. IEEE, 2009, pp. 339–348.

[47] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: an empirical study," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1383–1420, 2014.

[48] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1. IEEE, 2010, pp. 95–104.

[49] F. Thung, D. Lo, and L. Jiang, "Detecting similar applications with collaborative tagging," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 600–603.

[50] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *2013 20th Working conference on reverse engineering (WCRE)*. IEEE, 2013, pp. 182–191.

[51] N. Chen, S. C. Hoi, S. Li, and X. Xiao, "Simapp: A framework for detecting similar mobile applications by online kernel learning," in *Proceedings of the eighth ACM international conference on web search and data mining*, 2015, pp. 305–314.

[52] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on github," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 13–23.