LEGION: Harnessing Pre-trained Language Models for GitHub Topic Recommendations with Distribution-Balance Loss

Yen-Trang Dang
trang.dy190114@sis.hust.edu.vn
Hanoi University of Science and Technology
Hanoi, Vietnam

Thanh Le-Cong
congthanh.le@student.unimelb.edu.au
The University of Melbourne
Melbourne, Australia

Phuc-Thanh Nguyen
thanh.np200594@sis.hust.edu.vn
Hanoi University of Science and Technology, Vietnam
Hanoi, Vietnam

Anh M. T. Bui*
anhbtm@soict.hust.edu.vn
Hanoi University of Science and Technology, Vietnam
Hanoi, Vietnam

Phuong T. Nguyen
phuong.nguyen@univaq.it
Università degli studi dell'Aquila
67100 L'Aquila, Italy

Bach Le
bach.le@unimelb.edu.au
The University of Melbourne
Melbourne, Australia

Quyet-Thang Huynh
thanghq@soict.hust.edu.vn
Hanoi University of Science and Technology
Hanoi, Vietnam

Abstract

Open-source development has revolutionized the software industry by promoting collaboration, transparency, and community-driven innovation. Today, a vast amount of various kinds of open-source software, which form networks of repositories, is often hosted on GitHub – a popular software development platform. To enhance the discoverability of the repository networks, i.e., groups of similar repositories, GitHub introduced repository topics in 2017 that enable users to more easily explore relevant projects by type, technology, and more. It is thus crucial to accurately assign topics for each GitHub repository. Current methods for automatic topic recommendation rely heavily on TF-IDF for encoding textual data, presenting challenges in understanding semantic nuances.

This paper addresses the limitations of existing techniques by proposing Legion, a novel approach that leverages Pre-trained Language Models (PTMs) for recommending topics for GitHub repositories. The key novelty of Legion is three-fold. First, Legion leverages the extensive capabilities of PTMs in language understanding to capture contextual information and semantic meaning in GitHub repositories. Second, Legion overcomes the challenge of long-tailed distribution, which results in a bias toward popular topics in PTMs, by proposing a Distribution-Balanced Loss (DB Loss) to better train the PTMs. Third, Legion employs a filter to eliminate vague recommendations, thereby improving the precision of PTMs. Our empirical evaluation on a benchmark dataset of real-world GitHub repositories shows that Legion can improve vanilla PTMs by up to 26% on recommending GitHubs topics. Legion also can suggest GitHub topics more precisely and effectively than the state-of-the-art baseline with an average improvement of 20% and 5% in terms of Precision and F1-score, respectively.

*Anh M. T. Bui is the corresponding author

EASE 2024, 18–21 June, 2024, Salerno, Italy

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Software and its engineering → Software libraries and repositories**;
• **Computing methodologies → Machine learning**.

ACM Reference Format:
Yen-Trang Dang, Thanh Le-Cong, Phuc-Thanh Nguyen, Anh M. T. Bui, Phuong T. Nguyen, Bach Le, and Quyet-Thang Huynh. 2024. LEGION: Harnessing Pre-trained Language Models for GitHub Topic Recommendations with Distribution-Balance Loss. In *Proceedings of The 28th International Conference on Evaluation and Assessment in Software Engineering (EASE 2024).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Open-source development has fundamentally transformed the landscape of the software industry, championing principles of collaboration, transparency, and community-driven innovation. The imperative for developers to engage in exploration, sharing, and collaboration within the realm of open-source software (OSS) highlights the significance of a dedicated platform. Among the existing platforms, GitHub stands out as the foremost choice for hosting Git repositories, establishing itself as a cornerstone in the open-source ecosystem. This platform serves as a centralized hub where developers can seamlessly host, review, and manage their code repositories [7, 38, 46]. To foster discoverability and contribution to related projects, GitHub introduced tags – a pivotal feature in 2017 – allowing developers to tag repositories with relevant topics [13].

Topics convey salient descriptions of a repository, providing insights into various aspects such as the project's goals, evolving features, and technical details encompassing libraries and frameworks employed [42, 53]. Assigning the right topics to a GitHub repository is crucial for enhancing its discoverability among developers. It serves as a bridge between the social and technical aspects of repositories, potentially attracting interested users [44, 45, 51, 56]. On the contrary, an inaccurate assignment of topics compromises the usefulness of the GitHub topics and hinders the seamless recommendation of repositories to potential users.

Table 1: F1-score of well-known Pretrained Language Models on Top-1 prediction for "Head," "Mid," and "Tail" labels.

	Head	Mid	Tail
BERT	0.409	0.08	0.00
ELECTRA	0.358	0.00	0.00

To tackle these issues, GitHub introduced Repo-Topix, an information retrieval-based recommender system to suggest suitable topics for each GitHub repository [14]. Following this, new techniques [11, 12, 23, 53] have been proposed to automatically recommend topics for GitHub repositories, leveraging textual data such as README files and descriptions. These approaches model the topic recommendation problem as either multi-class or multi-label classification, employing machine learning models to classify suitable topics for GitHub repositories based on labeled data. More specifically, previous studies encode the textual data of a GitHub repository into numerical vectors using TF-IDF and then employ classification models such as Logistic Regression [23], Multinomial Naïve Bayesian [12] or ZestXML [53] to assign relevant topics to the repository.

While current techniques demonstrate encouraging performance, there are aspects that need improvements. Specifically, existing methods mainly depend on TF-IDF for representing the textual data of GitHub repositories. While TF-IDF can capture the statistics of common terms in the textual data, it inherently lacks the ability to capture contextual information or semantic meaning of words. This limitation poses a challenge in **understanding the underlying semantics of textual data (Challenge C1)** for accurately tagging repositories. To tackle this challenge, a potential solution is to harness Pre-trained Language Models (PTMs), which demonstrated remarkable language understanding capabilities due to their effective training on vast amounts of textual data.

However, employing PTMs for GitHub Topic Recommendation proves to be non-trivial. Izadi et al. [23] discovered that DistillBERT [41] is less effective than TF-IDF + LR [23]. We replicated their experiments with four well-known PTMs and found that these models are also less accurate compared to state-of-the-art techniques such as TF-IDF + LR [23] and TF-IDF + ZestXML [53], thus confirming Izadi et al.'s findings. In an in-depth analysis, we attribute this phenomenon to the **long-tailed distribution of GitHub topics (Challenge C2)**, in which a small proportion of topics is associated with a vast number of repositories, while a large number of other topics are assigned to very few repositories [59]. Consequently, PTMs exhibit a bias toward popular topics, preferring highly frequent labels to rare but correct/useful labels. Indeed, in our initial experiments (Table 1), while PTMs perform well in head (i.e., most frequent) topics (appearing in at least 30 repositories) with F1-score around 0.4, their effectiveness significantly drops to nearly zero on the mid and tail topics, i.e., topics with frequencies ranging from 30 to 9 and falling below 9. Consequently, we found that these PTMs perform worse than state-of-the-art baselines on recommending GitHub topics (see Section 5.1).

To address the aforementioned issues, we introduce Legion (LanguagE Models for GItHub TOpic RecommendatioN), an approach that effectively fine-tunes PTMs in GitHub Topic recommendation with Distribution-Balanced Loss (DB Loss) [55]. DB Loss tackles long-tailed distribution in multi-label classification problems by integrating re-sampling, re-balanced weighting, and negative tolerant regularization. In particular, it first employs class-aware sampling which samples training examples such that different class has a similar amount of training examples in each epoch. Then, it leverages re-balanced weights to reduce redundant information of label co-occurrence, caused by re-sampling. Finally, it reduces the bias from "easy-to-classify" negative instances by explicitly assigning lower weight to them. Additionally, during the inference stage, we introduce a filter to eliminate low-confident recommendations, thereby enhancing the precision of predictions. In summary, the benefit of Legion is three-fold. First, leveraging PTMs allows Legion to extract the underlying semantics of textual data in GitHub repositories, addressing Challenge C1. Second, DB Loss allows Legion to handle popularity bias caused by the long-tailed distribution of GitHub topics, thereby addressing Challenge C2. Third, the low-confident filter enhances the precision of Legion.

We evaluated Legion on a dataset of 15,262 repositories crawled from GitHub with 665 unique labels. Evaluation results showed that Legion can improve vanilla PTMs by up to 26% on recommending GitHub topics. Legion also can suggest GitHub topics more precisely and effectively than state-of-the-art topics with an average improvement of 20% and 5% in terms of Precision and F1-score, respectively.

Contributions. In summary, we make the following contributions:

- **Investigation.** We empirically investigate the impact of long-tailed distribution of GitHub topics on Pre-trained Language Models (PTM). Our experimental results reveal the lack of effectiveness of PTMs fine-tuned with commonly-used Binary Cross Entropy Loss on low frequency GitHub topics with nearly zero F1-scores.
- **Solution.** We introduce Legion, a novel recommendation technique that leverages PTM for suggesting relevant topics for GitHub repositories based on their textual data. Our proposed solution incorporates a Distribution-Balanced Loss and a Low-Confident Filter for effective training and precise inference of PTMs.
- **Evaluation.** We empirically evaluate the performance of Legion and its effectiveness in enhancing PTMs. Our experiments also show that Legion can significantly improve PTMs, outperforming the best baseline by 20% and 5% over in terms of Precision and F1-score.

Data Availability. To support the open science initiative, we publish a replication package including our dataset and trained models [8]. +We also provide an open-source implementation of Legion at

https://github.com/RISE-BKAI/LEGION/

2 BACKGROUND AND RELATED WORK

This section introduces a brief overview of the background and related works. First, we present recent studies on the problem of topic recommendation for GitHub repositories. Subsequently, we delve into Large Language Models (LLMs) and their applications for Software Engineering Tasks.

2.1 Topic Recommendation for GitHub Repositories

GitHub topic recommendation is related to suggesting relevant topics to repositories on GitHub. In this section, we first recall state-of-the-art techniques for this topic, and then introduce our problem formulation.

2.1.1 State-of-the-arts. Recent approaches in GitHub topic recommendation falls into two main categories.

Probability-based Recommendation: Recommenders based on probability [10–12], formulate this problem as *multi-class classification* and then employ algorithms, such as Multinomial Naïve Bayesian [12] to output probability that indicate how likely a topic belong to GitHub repository. Particularly, given these probabilities, these techniques rank topics to offer recommendations. In this work, we exclude probability-based techniques such as MNBN [12] as they have been proven to be less effective than the multi-label-based approaches described below.

Multi-label Classification: In contrast to the aforementioned approaches, Izadi et al. [23] recently formalized the GitHub repository topic recommendation task as a *multi-label classification* problem [19]. They conducted empirical studies exploring various combinations of (1) document representation and (2) multi-label classification. Their experiments highlighted

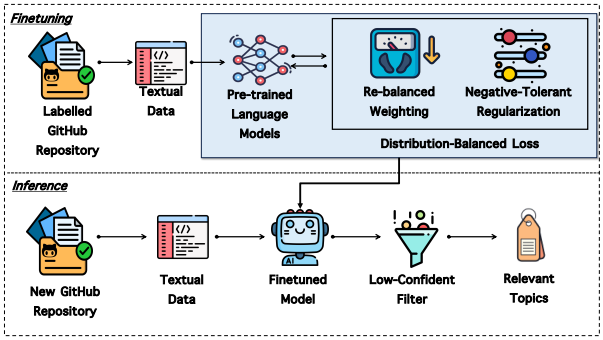

Figure 1: Overall workflow of LEGION.

Logistic Regression with TF-IDF embeddings as the superior combination for topic recommendation, outperforming probability-based recommendation approaches [23]. Widyasari et al. [53] further explores this direction by considering user-defined topics, formulating this problem as an extreme multi-label classification problem. Then, they conducted an exploration study to examine the effectiveness of XML techniques and found that ZestXML [17] is the best-performing approach in suggesting GitHub topics.

2.1.2 Problem Formulation. In this work, we focus on GitHub featured topics, which are carefully designed and maintained by GitHub and its community, following Izadi et al. [23] as user-defined GitHub topics may be noisy and of low quality. However, as rare topics may be still correct and important, we follow Widyasari [53] to not remove them. This allows us to ensure both the quality of our dataset and the reliability of the recommendation systems produced in this study. As a result, we obtained a dataset of 665 unique topics (see Section 4.2). We formulate our task as a multi-label classification problem, which takes as input a set of Github's featured topics $\mathcal{T}$ and a new (unlabelled) Github repository $\mathcal{G}$ and outputs a set of labels $\mathcal{T}_{new} \subseteq \mathcal{T}$ as topics of $\mathcal{G}$.

2.2 Pre-trained Language Models for Software Engineering Tasks

Pre-trained Language Models (PTMs) [6, 9, 27, 63] become popular in many domains including Natural Language Processing and Software Engineering thanks to their remarkable capabilities. These models are commonly built based on a Transformer architecture [47] and trained on a massive amount of data, allowing them to learn and capture both the syntax and semantics of human language. Given these capabilities, PTMs showed excellent performance in extracting semantic features from textual data and great promise to be fine-tuned for tasks that they were not initially trained for [20, 36].

Inspired by these successes, PTMs have been widely adopted in Software Engineering tasks such as code generation [2, 31, 52], program analysis [4, 21, 24], and program understanding/reasoning [1, 25, 48, 60]. Among these tasks, the applications of PTMs for Natural Language-based Software Engineering (NLBSE) tasks are most closely to our study. Zhang et al. [58] investigated the capabilities of PTMs on sentiment analysis for Software Engineering. He et al. [18] showed the potential of PTMs for recommending tags in Stack Overflow posts. Wang et al. [49] proposed the use of PTMs for labeling GitHub issues. Messaoud [35] applied PTMs for detecting duplicated bug reports. PTMs have been also applied to various other NLBSE tasks such as requirement classification [32], README simplication [15], Software Q&A [57]. Different from these works, our work delves into the problem of GitHub Topic Recommendation. We also investigate challenges, e.g., long-tailed distributions for applying PTMs on this task, and propose effective mechanisms for handling these challenges.

3 PROPOSED SOLUTION

Figure 1 illustrates the overall workflow of LEGION in both the fine-tuning and inference phases. First, in the fine-tuning phase, we fine-tune Pre-trained Models such as BERT [9] and RoBERTa [63] on labeled GitHub repositories (see the considered PTMs in Section 3.4). These PTMs are fine-tuned using Distribution-Balanced Loss (see Section 3.2) to obtain the optimized models. Next, in the inference phase, these PTMs can be used to recommend topics for new GitHub repositories. Finally, to ensure the precision of suggested topics, we also use a filter to prune low-confident predictions (see details in Section 3.3).

3.1 Data Pre-processing

3.1.1 Textual Data. Regarding textual data, we follow prior works [23, 53] to construct the data from related documents including README files and descriptions. We also perform several cleaning steps including removing alphanumeric characters following Widyasari et al. [53]. The details of the cleaning step can be found in their replication package.[1]

3.1.2 Topics. Regarding GitHub topics, the manual creation of most topic tags, many of them carry spelling mistakes or ambiguous meanings that render them unusable. Therefore, to ensure the quality of our dataset, we focus on GitHub's featured topics, which are carefully designed and maintained by GitHub and its community. The full list of such topics, as well as their alias, can be found on GitHub's explore repository.[2] We removed all labels that do not appear in the aforementioned list, and further augment the data by converting alias labels to their corresponding featured label. Note that, different from Izadi et al. [23], after filtering out non-featured topics from [53], we also did not omit any low-frequency topics. As a result, we observe a phenomenon of long-tail distribution even among featured topics, with up to 33.6% of topics appearing less than 4 times.

3.2 Distribution-Balanced Loss

Binary Cross Entropy (BCE) is a standard and commonly-used loss function for finetuning Pre-trained Language Models in prior works [18, 24, 30, 58]. Unfortunately, our experimental results show that BCE has a negative impact on the performance of PTMs, especially in less frequent GitHub topics (see details in Section 5.1). Therefore, we propose to use Distribution-Balanced Loss (DBLoss) [55] for effectively finetuning PTMs. DBLoss consists of three components: (1) Re-sampling, (2) Re-Balanced Weighting, and (3) Negative-Tolerant Regularization. These components are described in detail as follows.

3.2.1 Re-sampling. For fine-tuning a deep learning model in a supervision manner, it is necessary to sample examples from training data. The most common approach is to select these examples randomly with equal probability. However, to deal with the imbalance nature of long-tailed distribution, DBLoss leverages a popular strategy, namely class-aware sampling [43]. More specifically, it first uniformly selects a class from the entire set of classes, i.e., topics, and subsequently randomly picks an example from the chosen class. This process iterates throughout each training epoch with pre-defined number of times for each class visited. Typically, the number is defined as the maximum number of training example for a class. More formally,

$$N_e = \max(n_1, \ldots, n_C) \quad (1)$$

where, N_e is the number of times for each class visited in an epoch e, $C = |\mathcal{T}|$ is the number of classes, i.e., featured topics, and n_i is the number of examples for class i. In case of significant imbalance, N_e can be reduced to control the data scale within one epoch.

[1] https://figshare.com/s/dc6d69629442c6ac3bbb
[2] https://github.com/github/explore

3.2.2 Re-balanced Weighting. While re-sampling can partially mitigate the impact of imbalance in long-tailed distribution, prior work [55] shows that it can induce inner-class imbalance and may even exaggerate the inter-class imbalance. Therefore, DBLoss leverages a re-balanced weighting strategy based on the expectation of Class-level sampling frequency P_i^C and Instance-level sampling frequency $P^I(x^k)$ (k is an instance) to mitigate the extra imbalance caused by re-sampling. Particularly, it first estimates the expectation of these sampling frequencies as follows:

$$P_i^C = \frac{1}{C}\frac{1}{n_i} \tag{2}$$

$$P^I\left(x^k\right) = \frac{1}{C}\sum_{y_i^k=1}\frac{1}{n_i} \tag{3}$$

where x^k is an instance,i.e., a training example, $y_i^k \in \{0,1\}$ denotes if sample k belong to class i or not, $C = |\mathcal{T}|$ is the number of classes, i.e., featured topics, n_i is the number of examples for class i.

Then, DBLoss defines a re-balancing weight r_i^k for an instance x^k as follows:

$$r_i^k = \frac{P_i^C\left(x^k\right)}{P^I\left(x^k\right)} \tag{4}$$

This weight allows DBLoss to close the gap between expected sampling times and actual sampling times during the training stage, thus reducing the imbalance issue. Finally, to make the optimization process stable, it uses a smoothing function for mapping the weight to the expected value ranges. Particularly, the smoothed weight is calculated by:

$$\hat{r}_i^k = \alpha + \frac{1}{1+\exp\left(-\beta\times\left(r_i^k-\mu\right)\right)} \tag{5}$$

Given this weight, DBLoss defines Re-balanced Binary Cross Entropy Loss, which will be used together with Negative-Tolerant Regularization to form the final loss as follows:

$$\begin{aligned}\mathcal{L}_{R-BCE}\left(x^k,y^k\right) = \quad & \frac{1}{C}\sum_{i=0}^{C}\Bigg[y_i^k\log\left(1+e^{-z_i^k}\right)\\ & +\left(1-y_i^k\right)\log\left(1+e^{z_i^k}\right)\Bigg]\times\hat{r}_i^k\end{aligned} \tag{6}$$

where (x^k, y^k) is an training example, z^k denotes the output of a classifier and $\hat{r}_i^k$ is the smoothed re-balanced weight calculate by Equation 5.

3.2.3 Negative-Tolerant Regularization. Next, the domination of negative classes in multi-label classification may introduce the problem of over-suppression, in which deep learning models trained by Binary Cross Entropy loss tend to be biased by negative classes. Particularly, less frequent classes, i.e., topics, could have limited positive samples and a huge number of negative ones, making them tend to provide lower output probability and thus, a negative prediction. To address this issue, DBLoss aims to provide a sharp drop in the loss created by negative prediction once it is optimized to be lower than a certain threshold. This allows DBLoss to avoid the suppression from these predictions. Particularly, it uses a Negative-Tolerant Regularization, which employs a non-zero bias initialization to serve as thresholds, followed by the linear scaling to the negative logits. More formally, the regularization is calculated by:

$$\begin{aligned}\mathcal{L}_{NT-BCE}\left(x^k,y^k\right) = \quad & \frac{1}{C}\sum_{i=0}^{C}y_i^k\log\left(1+e^{-\left(z_i^k-v_i\right)}\right)\\ & +\frac{1}{\lambda}\left(1-y_i^k\right)\log\left(1+e^{\lambda\left(z_i^k-v_i\right)}\right)\end{aligned} \tag{7}$$

where (x^k, y^k) is an training example, z^k denotes the output of a classifier, v_i is a class-specific bias, and λ denotes the scale factor that control the regularization.

3.2.4 Final Loss. Finally, these aforementioned components are integrated into the final Distribution-Balanced Loss as follows:

$$\begin{aligned}\mathcal{L}_{DB}\left(x^k,y^k\right) = \quad & \frac{1}{C}\sum_{i=0}^{C}\hat{r}_i^k\Bigg[y_i^k\log\left(1+e^{-\left(z_i^k-v_i\right)}\right)\\ & +\frac{1}{\lambda}\left(1-y_i^k\right)\log\left(1+e^{\lambda\left(z_i^k-v_i\right)}\right)\Bigg]\end{aligned} \tag{8}$$

3.3 Filtering Low-Confident Recommendations

In GitHub Topic Recommendation techniques, it is a common approach to suggest a set of top-k predictions for users. However, we have observed that these techniques may not always exhibit confidence in their predictions, as indicated by low output probabilities. This lack of confidence can adversely affect the precision of the systems. In a recommendation system, high precision is crucial, as imprecise recommendations (false positives) can reduce the trust of users [5, 26, 40].

To address this issue, we propose a filter to eliminate low-confidence predictions from the set of top-k predictions. In other words, we advocate for a refinement process that considers the reliability of each prediction. Formally, the output predictions of a model can be defined as follows:

$$O = \{p \in O_k^{origin} | \mathcal{P}(p) \geq \tau\} \tag{9}$$

where, O_k^{origin} represents a set of k predictions generated by a model, $\mathcal{P}(p)$ denotes the output probability associated with a particular prediction, and τ serves as a filtering threshold. We consider τ to be a hyper-parameter, which will be tuned in a validation set to obtain optimal value.

3.4 Pre-trained Language Models

Theoretically, Legion's methodology including Distribution-Balanced Loss and Low-Confident Filter can be applied for fine-tuning and inference of any Pre-trained Language Models. However, given resource constraints, our work focuses on four well-known PTMs including BERT [9], BART [27], RoBERTa [63], and ELECTRA [6]. This section recalls these models in detail.

3.4.1 BERT. (Bidirectional Encoder Representations from Transformers) is a language model developed by Google. It utilizes a bidirectional architecture to understand the context of words by considering both left and right surrounding words simultaneously. The model's training procedure includes two steps: pre-training and fine-tuning. It was pre-trained on a diverse range of datasets, a combination of the BookCorpus [62] plus English Wikipedia, with two objectives: Masked language modeling (MLM) and Next sentence prediction (NSP). In this study, we focus on the original base model of BERT with 110 millions of parameters.

3.4.2 BART. is a denoising autoencoder built with a sequence-to-sequence model developed by Facebook. It is a transformer encoder-decoder with a bidirectional (BERT [9]-like) encoder and an auto-regressive (GPT [39]-like) decoder. BART is pre-trained by corrupting text with an arbitrary noising function and learning a model to reconstruct the original text. The architecture is closely related to that used in BERT, with some differences, and in total, it contains roughly 10% more parameters than the equivalently sized BERT model.

3.4.3 RoBERTa. (Robustly optimized BERT approach) an improved recipe for training BERT [9] model, that can match or exceed the performance of all of the post-BERT methods. RoBERTa is trained with dynamic masking, full sentences without NSP loss, large mini-batches, and a larger byte-level BPE. The dataset for training includes five English-language corpora of

Table 2: Detailed statistics of "Head," "Mid," and "Tail" labels.

	Head	Mid	Tail	Total
# Labels	208	215	242	665
# Repositories	14,667	3,354	545	15,262

varying sizes and domains including English Wikipedia, OpenWebText,[3] BookCorpus [62], and CC-News [34].

3.4.4 ELECTRA. is a new method for self-supervised language representation learning with little amount of computation. Its models are trained to distinguish "real" input tokens vs. "fake" input tokens generated by another neural network, similar to the discriminator of a GAN [16]. ELECTRA includes two neural networks called a generator and a discriminator. Despite the similar structure to GAN, the generator, a small masked language model, is trained with maximum likelihood rather than adversarially. After pre-training, the generator is thrown out and the discriminator is fine-tuned on downstream tasks.

4 EMPIRICAL SETTINGS

4.1 Research Questions

Our empirical evaluation aims to answer the following research questions:

RQ$_1$: *What is the impact of the long-tailed distribution of GitHub topics on Pre-trained Language Models?* This research question pertains to the impact of the long-tailed distribution on the effectiveness of Pre-trained Language Models (PTMs) in recommending GitHub topics. Particularly, we evaluate four well-known PTMs including BERT [9], RoBERTa [63], BART [27], and ELECTRA [6] and compare their effectiveness with a state-of-the-art baseline, namely ZestXML [53] and LR [23]. To assess the models' performance across labels with varying frequencies, we adopt the approach proposed by Huang et al. [22] and Zhou et al. [61] to partition the labels into roughly equal subsets, namely **head**, **mid**, and **tail**. **Head** labels denote the most frequently occurring ones, **tail** labels represent the least frequent, and **mid** labels encompass the remaining ones. Detailed statistics of these labels are presented in Table 2.

RQ$_2$: *Is Legion effective in improving Pre-trained Language Models on GitHub Topic Recommendation?* In this question, we investigate the impact of employing Legion to enhance the performance of distinct Pre-trained Language Models. Going into more detail, we apply our methodology to four aforementioned PTMs (i.e., BERT [9], RoBERTa [63], BART [27] and ELECTRA [6]) and compare their effectiveness to the original models, which is fine-tuned following a standard loss, i.e., Binary Cross-Entropy [50].

RQ$_3$: *How effective is Legion compared to state-of-the-art baselines on recommending GitHub topics?* This research question delves into assessing the capability of Legion to recommend relevant topics for GitHub repositories. We conduct experiments on a dataset comprising 15,262 GitHub repositories, which was previously assembled by Widyasari et al. [53] (see Section 4.2) in terms of Precision, Recall and F1-Score (see Section 4.3). We compare our proposed approach against two baseline methods including TF-IDF+LR [23] and TF-IDF+ZestXML [53] (see Section 4.4).

RQ$_4$: *Which components of Legion contribute to its effectiveness?* To mitigate the impact of long-tailed distribution on Pre-trained Language Models, Legion uses two mechanisms, i.e., Distribution Loss for training and Low-Confident Filter for inference. This research question investigates the contribution of each mechanism in an ablation study by dropping them one by one and observing the change in Legion 's performance.

[3] https://github.com/jcpeterson/openwebtext

Table 3: Training, validation, and testing datasets.

Dataset	# GitHub Repositories	# Unique Topics
Training	11,282	638
Validation	1,000	363
Testing	2,980	507

4.2 Dataset

We started with data provided by a prior work [53], which has already been split into train and test sets of 17,018 and 4,225 repositories, respectively. As stated in Section 3.1, to ensure the quality of our dataset, we focus on GitHub's featured topics. The result is a dataset with total 15,262 repositories and 665 unique labels. We extract randomly 1,000 repositories from the training data to create a validation set. The dataset now contains 11,282 training, 2,980 testing and 1,000 validation repositories. The statistics of our training and testing datasets are shown in Table 3.

4.3 Evaluation Metrics

The experiments aim to evaluate the effectiveness of our proposed approach and state-of-the-art baselines in GitHub topic recommendation using three evaluation metrics, i.e., Precision@K, Recall@K, and F1-score@K, which have been widely utilized for the same purpose [33, 53]. Particularly, given the top-K predictions for a repository, all metrics are calculated globally based on the total count of true positives, false negatives, false positives defined as follows:

- **True Positives (TP):** a topic is suggested to a repository by a recommender, and it is an actual topic of the repository.
- **False Positives (FP):** a topic is suggested to a repository by a recommender, but it does not belong to the repository.
- **True Negatives (FP):** a topic is not suggested to a repository by a recommender, and it does not belong to the repository.
- **False Negatives (TP):** a topic is not suggested to a repository by a recommender, but it is an actual topic of the repository.

Precision measures how *precise/accurate* the suggested topics are. It is calculated by the ratio of true topics (true positives) over the total number of predictions, i.e., $P = \frac{TP}{TP + FP}$.

Recall measure how *complete* are suggested topics. It is calculated by the ratio of true topics suggested (true positives) by a recommendation system over the total number of actual topics. $R = \frac{TP}{TP + FN}$.

F1-score is a harmonic mean of the precision and recall, seeking the balance between these metrics, i.e., $F1 = \frac{2 \times (P \times R)}{P + R}$. In this work, we leverage Micro-F1 [29, 54], which takes label imbalance into account.

4.4 Baselines

To evaluate the effectiveness of Legion, we compare our approach with the following baselines:

- **TF-IDF + LR** [23]: After removing all labels with occurrences count smaller than 100, this method formulates the Github topic recommendation problem as a multi-label classification task. Textual inputs, including repositories's README and description, are encoded into TF-IDF vectors then passed through a Logistic Regression classifier. As part of the same study, a DistilBERT classifier was constructed but fell short compared to the TF-IDF+LR model. In our experiments, we refer to this approach as **LR**.
- **TF-IDF + ZestXML** [53]: This study emphasizes the importance of having a diverse label space, including rare and emerging topics, and thus, did not remove any low-frequency labels. Similar to the above approach, repositories' README and descriptions are preprocessed

Table 4: Effectiveness of PTMs, including BERT, BART, RoBERTa, and ELECTRA, compared to state-of-the-art baselines on different parts of GitHub topics' distribution.

Model	Head			Mid			Tail			All			Avg F1
	F@1	F@3	F@5	F@1	F@3	F@5	F@1	F@3	F@5	F@1	F@3	F@5	
BERT	0.409	0.505	0.504	0.081	0.166	0.180	0.0	0.0	0.0	0.374	0.474	0.475	0.441
BART	0.416	0.514	**0.511**	0.049	0.142	0.158	0.0	0.0	0.0	0.378	0.480	0.480	0.446
RoBERTa	0.366	0.441	0.445	0.0	0.0	0.0	0.0	0.0	0.0	0.329	0.405	0.410	0.381
ELECTRA	0.358	0.426	0.421	0.0	0.014	0.020	0.0	0.0	0.0	0.322	0.392	0.389	0.368
ZestXML	0.398	0.469	0.418	**0.235**	**0.444**	0.430	**0.169**	**0.291**	**0.257**	0.379	0.465	0.416	0.420
LR	**0.417**	**0.524**	0.443	0.181	0.362	**0.463**	0.028	0.028	0.028	**0.388**	**0.507**	**0.500**	**0.465**

and encoded into TF-IDF vectors. These vectors are classified using a Zero-shot XML algorithm called ZestXML. In our experiments, we refer to this approach as **ZestXML**.

- **Finetuned PTMs + BCE**: Pretrained Language Models have been proven to be useful in numerous tasks that require universal language understanding. Such PTMs can be adapted for the text classification tasks via the addition a fully-connected layer to produce output probability [22].
- **Finetuned PTMs + Focal Loss**: In addition to fine-tuning Pretrained Models (PTMs) using Binary Cross Entropy (BCE) Loss, we also incorporate Focal Loss [28] for fine-tuning. Focal Loss is a recognized loss function designed for imbalanced data. This loss strategy emphasizes training on a selective set of challenging examples and mitigate the impact of numerous easy negatives by assigning explicit weight to BCE Loss. In our experiments, we refer to this approach as **FL**.

4.5 Implementation Details

The AutoModelForSequenceClassification backbone[4] in transformers library was used to fine-tune our PTMs. The parameters for BERT [9] were initialized using the bert-base-uncased pretrained model. For RoBERTa [63], BART [27], and ELECTRA [6], the parameteres were initialized from roberta-base, bart-base, and electra-base-discriminator models. All of the above models can be found on HuggingFace.[5] Among them, bert-based-uncased and electra-base-discriminator follow the base BERT architecture with 110M parameters, while bart-base and roberta-base have 139M and 125M parameters respectively. The PTMs are fine-tuned using one Tesla V100 GPU which takes between 6 to 10 minutes for one epoch. The input sequences are truncated or padded to a maximum length of 512 and batched with batch size 32. The optimizer is AdamW with a weight decay of 0.01. The default learning rate for BCE models is 1e-5, but for models using DB loss the learning rate is 1e-4. Others parameters of both BCE and DB loss functions follow existing work [22, 55]. All the experiments were carried out with PyTorch. For the older baselines, TF-IDF+LR and ZestXML+LR, we reused the original preprocessing, training and inference code [53]. During the evaluation, we acquired the top-k predictions on test set, and calculate precision score, recall score, and micro F1 score using scikit-learn's functions [37].

5 EMPIRICAL RESULTS

This section reports and analyzes the experimental results by answering the research questions introduced in Section 4.1.

[4]https://huggingface.co/transformers/v3.0.2/model_doc/auto.html
[5]https://huggingface.co/models

5.1 RQ$_1$: *What is the impact of the long-tailed distribution of GitHub topics on Pre-trained Language Models?*

To investigate the impact of long-tailed distribution of GitHub topics, we evaluate the effectiveness of different PTMs, including BERT [9], RoBERTa [63], BART [27] and ELECTRA [6] on head, mid and tail labels. The experimental results are shown in Table 4.

Overall, PTMs show good performance when predicting head (i.e., most frequent) labels. However, a remarkable decline in performance becomes apparent when tasked with predicting mid and tail labels, characterized by lower frequencies. Specifically, all PTMs consistently achieve an F1-score within the range of 0.3 to 0.5 for head labels, whereas this performance drops to under 0.18 for mid subsets, even under the most optimistic scenarios. Notably, all of the PTMs considered in our study could not predict correctly any tail labels, i.e., all the corresponding F@1, F@3, and F@5 scores are equal to 0. This may be attributed to potential overfitting on head labels and the lack of training data on the less frequent labels.

The results show that PTMs perform much worse compared to state-of-the-art baselines, even on the head labels. Particularly, the best baseline, i.e., LR, outperforms PTMs in F@1 and F@3 for head labels, with PTMs only surpassing the baselines in F@5. The leading PTM, BERT-base, falls short when compared to both ZestXML (showing an average performance decrease of around 62% in F@1, F@3, and F@5) and LR (demonstrating an average decrease of 57% in F@1, F@3, and F@5). As a result, in full data, though showing comparable performance with ZestXML, PTMs consistently perform worse than LR in F@1, F@3, and F@5 by 4-26% on average.

These findings diverge from our initial assumptions regarding the semantic understanding capabilities of PTMs, possibly attributing to the imbalance in training data and the shortage of data in less frequent labels, stemming from the long-tailed distribution of GitHub topics. This distribution poses challenges for models when learning our specific tasks and introduces a popularity bias in PTMs. These observations highlight the necessity for new approaches to address the impact of long-tailed distributions for effectively applying PTMs on this task.

> ***Answer to RQ$_1$:*** *The long-tailed distribution of GitHub topics significantly affects the performance of Pre-trained Language Models, with nearly zero performance on less frequent topics. This leads to suboptimal performance for PTMs when compared to state-of-the-art techniques.*

5.2 RQ$_2$: *Is Legion effective in improving Pre-trained Language Models on GitHub Topic Recommendation?*

In this experiment, we investigate the effectiveness of Legion on enhancing Pretrained Language Models (PTMs) on GitHub Topic Recommendation. The detailed results are shown in Table 5. Overall, we can see that Legion can

Table 5: Effectiveness of LEGION on improving PTMs, on different parts of topics' distribution. BERT$_L$, RoBERTa$_L$, BART$_L$, ELECTRA$_L$ are the performance of improved version of BERT, BART, RoBERTa, and ELECTRA with LEGION, respectively.

Model	Head			Mid			Tail			All			Avg F1
	F@1	F@3	F@5	F@1	F@3	F@5	F@1	F@3	F@5	F@1	F@3	F@5	
BERT	0.409	0.505	0.504	0.081	0.166	0.180	0.0	0.0	0.0	0.374	0.474	0.475	0.441
BERT$_L$	0.432	0.532	0.535	0.363	0.467	0.474	0.051	0.063	0.079	0.421	0.521	0.525	0.489
Improvement	↑5.6%	↑5.3%	↑6.2%	↑348%	↑181%	↑165%	↑ N/A	↑ N/A	↑ N/A	↑12.6%	↑9.9%	↑10.5%	↑10.9%
BART	0.416	0.514	0.511	0.049	0.142	0.158	0.0	0.0	0.0	0.378	0.480	0.480	0.446
BART$_L$	0.431	0.533	0.540	0.315	0.458	0.466	0.050	0.096	0.126	0.414	0.521	0.529	0.488
Improvement	↑3.6%	↑3.7%	↑5.7%	↑543%	↑222%	294%	↑ N/A	↑ N/A	↑ N/A	↑9.5%	8.5%	↑10.2%	↑9.4%
RoBERTa	0.366	0.441	0.445	0.0	0.0	0.0	0.0	0.0	0.0	0.329	0.405	0.410	0.381
RoBERTa$_L$	0.430	0.527	0.530	0.310	0.425	0.435	0.045	0.045	0.045	0.412	0.513	0.517	0.481
Improvement	↑17.5%	↑19.5%	↑19.1%	↑ N/A	↑ N/A	↑ N/A	↑ N/A	↑ N/A	↑ N/A	↑25.2%	↑26.7%	↑26.1%	↑26.0%
ELECTRA	0.358	0.426	0.421	0	0.014	0.020	0.0	0.0	0.0	0.322	0.392	0.389	0.368
ELECTRA$_L$	0.375	0.440	0.437	0.205	0.266	0.267	0.017	0.032	0.032	0.354	0.419	0.417	0.397
Improvement	↑4.7%	↑3.3%	↑3.8%	↑ N/A	↑1,900%	↑1,335%	↑ N/A	↑ N/A	↑ N/A	↑9.9%	↑6.9%	↑7.2%	↑7.9%

substantially improve the performance of all PTMs by 7.9%, 9.4%, 10.9%, and 26% in terms of average F1-score on ELECTRA, BART, BERT, and RoBERTa, respectively. More specifically, the refined PTMs incorporating LEGION exhibit notable improvements over their original counterparts, ranging from 9.9% to 25.2%, 6.9% to 26.7%, and 7.2% to 26.1% in top-1, top-3, and top-5 predictions, respectively.

These improvements stem from the enhanced performance of PTMs on different subsets of labels including head, mid, and tail labels. Specifically, for head labels, LEGION yields substantial improvements in the prediction quality (reflected by F1-score) of BERT, BART, ELECTRA, and RoBERTa, ranging from 5.3% to 6.2%, 3.6% to 5.7%, 3.3% to 4.7%, and 17.5% to 19.5%, respectively. For mid-frequency labels, LEGION showcases its ability by aiding PTMs in achieving an F1-score of approximately 0.4, translating to an improvement of at least 2.5 times. Notably, the LEGION can improve the F1-score of RoBERTa and ELECTRA from nearly zero to a noteworthy range of 0.205 to 0.435. Furthermore, concerning tail labels, where the initial performance of PTMs is zero, the enhanced version of PTMs with LEGION exhibits an F1-score of up to 0.126. Although the performance remains modest, this improvement is noteworthy and encouraging, especially considering the subpar performance observed previously.

The experimental results demonstrate the effectiveness of LEGION in improving PTMs, especially in mid frequent labels. These findings highlight the usefulness of LEGION in mitigating the impact of the long-tailed distribution of GitHub topics on PTMs. Despite notable improvements, PTMs exhibit suboptimal performance on tail distribution. LEGION, while effective, may not fully address this challenge alone, revealing its limitations. To overcome this, a plausible strategy is to employ LEGION in conjunction with other techniques that excel in handling tail labels, such as ZestXML. We show that this synergistic approach can enhance overall performance and provide a more comprehensive solution, as explained in Section 6.1.

> ***Answer to RQ$_2$:*** *LEGION can significantly improve the performance of Pre-trained Language Models on recommending GitHub topics with improvements ranging from 7.9% to 26.0% in terms of F1.*

5.3 RQ$_3$: *How effective is LEGION compared to state-of-the-art baselines on recommending GitHub topics?*

To investigate the effectiveness of our approach, we compare LEGION with two state-of-the-art baselines including ZestXML [53] and LR [23]. The results obtained by the baselines and LEGION are shown in Table 6. Overall, LEGION outperforms both state-of-the-art baselines. Specifically, LEGION, achieves an F1-score of 0.489, a 5.2% and 16.4% increase over LR and ZestXML respectively. Our approach also consistently improves the best baseline by 8.5%, 2.8%, and 5.0% in top-1, top-3 and top-5 predictions, respectively.

Table 6: Effectiveness of our model compared to baselines in terms of Precision, Recall, and F1-score at top-k predictions with k from 1 to 5.

Model		ZestXML	LR	FL	LEGION	*Improvement*
Top-1	P	0.649	0.688	0.665	**0.744**	↑8.1%
	R	0.268	0.271	0.274	**0.293**	↑6.9%
	F	0.379	0.388	0.388	**0.421**	↑8.5%
Top-3	P	0.420	0.506	0.541	**0.616**	↑22.7%
	R	**0.520**	0.507	0.453	0.451	↓13.3%
	F1	0.465	0.507	0.493	**0.521**	↑2.8%
Top-5	P	0.308	0.440	0.523	**0.600**	↑36.4%
	R	**0.637**	0.577	0.472	0.467	↓26.7%
	F	0.416	0.500	0.497	**0.525**	↑5.0%
Avg	P	0.459	0.545	0.576	**0.653**	↑20.0%
	R	**0.475**	0.452	0.400	0.404	↓15.0%
	F1	0.420	0.465	0.459	**0.489**	↑5.2%

We observed that these improvements stem from the remarkable enhancement of LEGION's precision with an average increase of 20%. More specifically, our approach improves the most precise baseline by 8.1%, 22.7%, and 36.4% in top-1, top-3, and top-5 predictions, respectively. Regarding Recall, LEGION only outperforms state-of-the-art baseline in top-1 prediction with an increase of 6.9% while performing worse than them in Top-3 and Top-5 predictions by 13.3% and 26.7%. This can be attributed to the design of our method, which leverages a Low-Confident Filter to ensure the precision of the models. The design restricts the LEGION's prediction, resulting in the lower Recall. Without the filter, as seen in Table 8, LEGION will produce higher Recall with an increase of 17.7% and 34.3%, achieving comparable (and even better) performance with ZestXML and LR. However, the precision score is crucial as an imprecise recommendation, e.g., false positives, could lose the trust of users [5, 26, 40] and thus, is more important than Recall. Therefore, we believe that the filter is necessary for our approach.

Next, to understand how LEGION and the baselines deal with long-tailed distortion of GitHub topics, we also conduct an in-depth analysis of their performance in the three subsets: head, mid, and tail topics. Table 7 presents the detailed results of this analysis. Overall, LEGION is the best-performing

Table 7: Effectiveness of LEGION compared to state-of-the-art baselines on different parts of GitHub topics' distribution.

Model	Head			Mid			Tail			All			Avg F1
	F@1	F@3	F@5	F@1	F@3	F@5	F@1	F@3	F@5	F@1	F@3	F@5	
ZestXML	0.398	0.469	0.418	0.235	0.444	0.430	0.169	0.291	0.257	0.379	0.465	0.416	0.420
LR	0.417	0.524	0.443	0.181	0.362	0.463	0.028	0.028	0.028	0.388	0.507	0.500	0.465
LEGION	0.432	0.532	0.535	0.363	0.467	0.474	0.051	0.063	0.079	0.421	0.521	0.525	0.489
Improvement	↑3.6%	↑1.5%	↑20.7%	↑100.1%	↑29.0%	↑2.4%	↓30.2%	↓21.6%	↓30.7%	↑8.5%	↑2.8%	↑5.0%	↑5.2%

approach on the head and mid topics. Particularly, our approach shows improvements of up to 20.8% and 100.1% (in terms of F1-score) over the best baseline in head and mid labels, respectively. However, LEGION falls short of ZestXML in tail labels with a decrease of 21.6% - 30.7% in terms of F1-score. This is within our expectation as ZestXML is a zero-shot learning algorithm, which is designed specifically for rare and unseen label prediction. These results unveil the possibility of combining LEGION and ZestXML for better GitHub topic recommendation techniques. We discuss this possible combination and its challenge in Section 6.1.

Answer to RQ$_3$: *LEGION is effective in recommending GitHub topics, enhancing the state-of-the-art baseline by an average of 5.2% in F1. This improvement stems from more precise predictions, with a notable 20.0% enhancement in precision.*

5.4 RQ$_4$: *Which components of LEGION contribute to its effectiveness?*

In this research question, we perform an ablation study to analyze the components that account for the gain in performance. Table 8 depicts the obtained results from the corresponding experiments. LEGION$_{w/oLoss}$ and LEGION$_{w/oFilter}$ denote variants of LEGION without Distribution-Balanced Loss (BDLoss) and Low-Confident Filter, respectively.

Table 8: Ablation Study. LEGION$_{w/oLoss}$ and LEGION$_{w/oFilter}$ denotes variants of LEGION without DBLoss and Low-Confident Filtering, respectively.

Model		LEGION	LEGION$_{w/oLoss}$	LEGION$_{w/oFilter}$
Top-1	P	0.744	0.673(↓9.5%)	0.717 (↓3.6%)
	R	0.293	0.259 (↓11.6%)	0.297 (↑1.4%)
	F1	0.421	0.374 (↓11.2%)	0.420 (↓0.2%)
Top-3	P	0.616	0.546 (↓11.4%)	0.428 (↓30.5%)
	R	0.451	0.418 (↓7.3%)	0.531 (↑17.7%)
	F1	0.521	0.474 (↓9.0%)	0.474 (↓9.0%)
Top-5	P	0.600	0.526 (↓12.3%)	0.303 (↓49.5%)
	R	0.467	0.432 (↓7.5%)	0.627 (↑34.3%)
	F1	0.525	0.474 (↓9.7%)	0.408 (↓22.3%)
Average	P	0.653	0.582 (↓11.0%)	0.483 (↓26.1%)
	R	0.404	0.370 (↓8.4%)	0.485 (↑20.1%)
	F1	0.489	0.441 (↓9.9%)	0.434 (↓11.2%)

We can see that, without BDLoss, LEGION witnessed a consistent decrease in terms of Precision, Recall, and F1-score. Particularly, in terms of Precision, LEGION$_{w/oLoss}$ show a drop of 9.5%, 11.4% and 12.3% in top-1, top-3 and top-5 predictions, respectively, resulting a decrease of 11% on average. Meanwhile, the Recall of LEGION$_{w/oLoss}$ is reduced by 8.4% on average over LEGION, with a decrease of 11.6%, 7.3% and 7.5% in top-1, top-3 and top-5 predictions, respectively. As a results, LEGION$_{w/oLoss}$ is less effective (in terms of F1-score) than original version by 9.9% with the decrease from 9.0% to 11.2% in different number of predictions.

We also can observe the similar trends in Precision and F1-score of LEGION$_{w/oFilter}$. Particularly, when the low-confident filter is left out, LEGION witnessed a 26.1% decrease in average precision than LEGION while observing a drop of 11.2% in terms of F1-score. However, we can see that LEGION has better recall without this filter, boosting the metrics between 1.4% to 34.3%. These results show that the filter may remove some low-confident yet correct predictions. As discussed in Secion 5.3, we believe that the precision of a recommendation is crucial so the filter is necessary for our approach. Nevertheless, this could be considered as a limitation of LEGION, we encourage future research to address this limitation by ensuring the high confidence for correct predictions.

Overall, our experimental results demonstrate that removing either component from LEGION causes a worse overall quality of predictions, reflected by the F1-score. This suggests that both DBLoss and Low-Confident Filter are important for LEGION to perform effectively. We also suggest future works to further improve the confident score of correct predictions for avoiding the losing in Recall by Low-Confident Filter.

Answer to RQ$_4$: *All components of LEGION contribute positively to its performance. Without Distribution-Balanced Loss and Low-Confident Filter, the performance of LEGION decreases by 9.9% and 11.2%.*

6 DISCUSSION

In this section, we discuss the possibility of a fusion between LEGION and ZestXML for better prediction. Afterward, we present the threats to the validity of our study.

6.1 Synergy of LEGION and existing techniques

As discussed in Section 5.2 and 5.3, we found that LEGION excels in head and mid labels but performs worse than ZestXML in tail labels, which rarely happened. In this section, we discuss the possibility of a combined approach of LEGION and existing approach for better GitHub topic recommendation. Particularly, we design a combined approach by combining top-3 predictions from ZestXML and top-5 predictions from LEGION for the final top-8 predictions. The rationale behind this design is: (1) we observe that LEGION and ZestXML show the optimal effectiveness on top-5 and top-3 predictions, respectively, and (2) we avoid bias on tail labels, which rarely happens and may harm performance. Table 9 illustrates the effectiveness of the synergistic approach compared to LEGION and ZestXML.

Table 9: The effectiveness (F1-score) of combined approach of ZestXML and LEGION.

	ZestXML	LEGION	Combined approach
Head	0.340	**0.536**	0.532
Mid	0.341	0.473	**0.489**
Tail	0.193	0.079	**0.267**
All	0.337	0.525	**0.526**

Overall, we can see that the combined approach outperforms ZestXML and LEGION in mid and tail labels and only shows a slight decrease in

performance in head labels. Consequently, the approach shows the best performance in whole labels. While the improvement is marginal, we can see that this combined approach is more reliable than ZestXML and Legion as it offers consistent performance over different subsets of labels. We encourage future works to explore this direction for better GitHub topic recommendation techniques.

6.2 Threats to Validity

6.2.1 Internal Validity. This threat refers to possible flaws in our experiments and implementations. We have carefully checked the correctness of our implementation. We also published our source code along with trained models [8] and presented detailed hyper-parameters for the training of considered models in Section 4.5. Given these materials, other research can validate our results and findings. Therefore, we believe that there are minimal threats from this issue.

6.2.2 Construct Validity. This threat concerns the suitability of our evaluation. A source of these threats may stem from our evaluation metric. To minimize this threat, we employ well-known metrics, which are commonly used by prior studies [11, 23, 33, 53] for multi-label classification, including Precision, Recall, and F1-score.

6.2.3 External Validity This threat relates to the generalizability of our findings. Our experiments are conducted on the same dataset as prior work [53], crawled from GitHub. This raises a concern about the external validity of our findings as they may not generalize beyond GitHub repositories outside our dataset. However, we believe this threat is minimal as the dataset is exhaustively crawled from GitHub and consists of a large number of data points, which ensure their diversity. Another potential threats to our external validity is our selection of Pre-trained Language Models (PTMs), which may raise a risk that our finding may not generalize beyond these PTMs. Due to resources constraints, we remark this as our limitation and leave the further investigations for future work.

7 CONCLUSION AND FUTURE WORK

In this work, we addressed the challenges in GitHub topic recommendation, particularly focusing on the limitations of existing methods that heavily rely on TF-IDF for representing textual data and the negative impact of long-tailed distribution of topics. Our investigation revealed the ineffectiveness of Pre-trained Language Models (PTMs) in handling the long-tailed distribution of GH topics. To overcome these challenges, we proposed Legion as a novel approach that fine-tunes PTMs with Distribution-Balanced Loss to mitigate popularity bias caused by long-tailed distribution. Moreover, to ensure the precision of Legion, we also propose a Low-Confident Filter to eliminate imprecise predictions. The evaluation showed that Legion significantly improved PTMs by up to 26% in recommending GH topics. Our experiments also demonstrate the effectiveness of Legion, achieving higher precision and F1-score than state-of-the-art baselines.

In the future, we plan to extend Legion vertically and horizontally. Firstly, we plan to further improve our empirical evaluation. Particularly, we aim to explore (1) the impact of more PTMs and their architectures on GH topic recommendations and (2) measure the effectiveness of our approach and existing techniques in a more comprehensive setting with more evaluation metrics such as MCC [3] and larger dataset for providing more insights about our study. Moreover, as README is only a small part of a GitHub repository, we also want to add more additional context such as source code, filename or description/topics of relevant projects for enhancing textual inputs. Additionally, we also aim to investigate the generalization of Legion to other collaborative platforms beyond GitHub such as GitLab or Bitbucket. Lastly, we want to explore the integration of user feedback and interactions to refine Legion that could enhance the user experience.

Acknowledgment

This research is funded by Hanoi University of Science and Technology (HUST) under project number T2023-PC-002.

References

[1] Toufique Ahmed and Premkumar Devanbu. 2022. Few-shot training LLMs for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–5.

[2] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[3] Davide Chicco, Matthijs J Warrens, and Giuseppe Jurman. 2021. The Matthews correlation coefficient (MCC) is more informative than Cohen's Kappa and Brier score in binary classification assessment. *Ieee Access* 9 (2021), 78368–78381.

[4] Yiu Wai Chow, Max Schäfer, and Michael Pradel. 2023. Beware of the unexpected: Bimodal taint analysis. *arXiv preprint arXiv:2301.10545* (2023).

[5] Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*. 332–343.

[6] Kevin Clark, Thang Luong, Quoc V. Le, and Christopher Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *ICLR*. https://openreview.net/pdf?id=r1xMH1BtvB

[7] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. 1277–1286.

[8] Yen-Trang Dang, Thanh Le-Cong, Phuc-Thanh Nguyen, Anh M. T. Bui, Phuong T. Nguyen, Bach Le, and Quyet-Thang Huynh. 2023. Artifacts: LEGION: Harnessing Pre-trained Language Models for GitHub Topic Recommendations with Distribution-Balance Loss. Figshare. https://figshare.com/s/6e01956fbfcd9b7ca6de

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.

[10] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong Nguyen, and Riccardo Rubei. 2020. Topfilter: an approach to recommend relevant github topics. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[11] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T Nguyen, and Riccardo Rubei. 2022. HybridRec: A recommender system for tagging GitHub repositories. *Applied Intelligence* (2022), 1–23.

[12] Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Phuong T Nguyen. 2020. A multinomial naïve bayesian (mnb) network to automatically recommend topics for github repositories. In *Proceedings of the Evaluation and Assessment in Software Engineering*. 71–80.

[13] Shay Frendt. 2019. Introducing topics. https://github.blog/2017-01-31-introducing-topics/

[14] Kavita Ganesan. 2017. Topic suggestions for millions of repositories. https://github.blog/2017-07-31-topics/

[15] Haoyu Gao, Christoph Treude, and Mansooreh Zahedi. 2023. Evaluating Transfer Learning for Simplifying GitHub READMEs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1548–1560.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[17] Nilesh Gupta, Sakina Bohra, Yashoteja Prabhu, Saurabh Purohit, and Manik Varma. 2021. Generalized Zero-Shot Extreme Multi-Label Learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 527–535. https://doi.org/10.1145/3447548.3467426

[18] Junda He, Bowen Xu, Zhou Yang, DongGyun Han, Chengran Yang, and David Lo. 2022. PTM4Tag: sharpening tag recommendation of stack overflow posts with pre-trained models. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 1–11.

[19] Francisco Herrera, Francisco Charte, Antonio J Rivera, and María J del Jesus. 2016. Multilabel classification. In *Multilabel Classification*. Springer, 17–31.

[20] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *arXiv preprint arXiv:2308.10620* (2023).

[21] Qing Huang, Zhiqiang Yuan, Zhenchang Xing, Xiwei Xu, Liming Zhu, and Qinghua Lu. 2022. Prompt-tuned code language model as a neural knowledge base for type inference in statically-typed partial code. In *Proceedings of the 37th*

IEEE/ACM International Conference on Automated Software Engineering. 1–13.

[22] Yi Huang, Buse Giledereli, Abdullatif Köksal, Arzucan Ozgur, and Elif Ozkirimli. 2021. Balancing Methods for Multi-label Text Classification with Long-Tailed Class Distribution. 8153–8161. https://doi.org/10.18653/v1/2021.emnlp-main.643

[23] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. 2021. Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering* 26, 5 (2021), 1–33. https://doi.org/10.1007/s10664-021-09976-2

[24] Thanh Le-Cong, Hong Jin Kang, Truong Giang Nguyen, Stefanus Agus Haryono, David Lo, Xuan-Bach D Le, and Quyet Thang Huynh. 2022. AutoPruner: transformer-based call graph pruning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 520–532.

[25] Thanh Le-Cong, Duc-Minh Luong, Xuan Bach D Le, David Lo, Nhat-Hoa Tran, Bui Quang-Huy, and Quyet-Thang Huynh. 2023. Invalidator: Automated patch correctness assessment via semantic and syntactic reasoning. *IEEE Transactions on Software Engineering* (2023).

[26] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E James Whitehead. 2013. Does bug prediction support human developers? findings from a google case study. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 372–381.

[27] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7871–7880.

[28] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[29] Zachary Chase Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. 2014. Thresholding Classifiers to Maximize F1 Score. arXiv:1402.1892 [stat.ML]

[30] Fang Liu, Ge Li, Yunfei Zhao, and Zhi Jin. 2020. Multi-task learning based pre-trained language model for code completion. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 473–485.

[31] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D Le, and David Lo. 2023. Refining ChatGPT-generated code: Characterizing and mitigating code quality issues. *arXiv preprint arXiv:2307.12596* (2023).

[32] Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. 2022. PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[33] Yunbo Lyu, Thanh Le-Cong, Hong Jin Kang, Ratnadira Widyasari, Zhipeng Zhao, Xuan-Bach D. Le, Ming Li, and David Lo. 2023. Chronos: Time-Aware Zero-Shot Identification of Libraries from Vulnerability Reports. (2023), 1033–1045. https://doi.org/10.1109/ICSE48619.2023.00094

[34] Joel Mackenzie, Rodger Benham, Matthias Petri, Johanne R Trippas, J Shane Culpepper, and Alistair Moffat. 2020. CC-News-En: A large English news corpus. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 3077–3084.

[35] Montassar Ben Messaoud, Asma Miladi, Ilyes Jenhani, Mohamed Wiem Mkaouer, and Lobna Ghadhab. 2022. Duplicate bug report detection using an attention-based neural language model. *IEEE Transactions on Reliability* (2022).

[36] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. 2023. Recent advances in natural language processing via large pre-trained language models: A survey. *Comput. Surveys* 56, 2 (2023), 1–40.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[38] Huilian Sophie Qiu, Alexander Nolte, Anita Brown, Alexander Serebrenik, and Bogdan Vasilescu. 2019. Going farther together: The impact of social capital on sustained participation in open source. In *2019 ieee/acm 41st International Conference on Eoftware Engineering (ICSE)*. IEEE, 688–699.

[39] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[40] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. 2018. Lessons from building static analysis tools at google. *Commun. ACM* 61, 4 (2018), 58–66.

[41] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[42] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. 2017. Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 314–319.

[43] Li Shen, Zhouchen Lin, and Qingming Huang. 2016. Relay backpropagation for effective learning of deep convolutional neural networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*. Springer, 467–482.

[44] Christoph Treude and Margaret-Anne Storey. 2009. How tagging helps bridge the gap between social and technical aspects in software development. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 12–22. https://doi.org/10.1109/ICSE.2009.5070504

[45] Christoph Treude and Margaret-Anne Storey. 2010. Work item tagging: Communicating concerns in collaborative software development. *IEEE Transactions on Software Engineering* 38, 1 (2010), 19–34.

[46] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software engineering (ICSE)*. 356–366.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[48] Deze Wang, Zhouyang Jia, Shanshan Li, Yue Yu, Yun Xiong, Wei Dong, and Xiangke Liao. 2022. Bridging pre-trained models and downstream tasks for source code understanding. In *Proceedings of the 44th International Conference on Software Engineering*. 287–298.

[49] Jun Wang, Xiaofang Zhang, and Lin Chen. 2021. How well do pre-trained contextual language representations recommend labels for GitHub issues? *Knowledge-Based Systems* 232 (2021), 107476.

[50] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. 2022. A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science* 9, 2 (April 2022), 187–212. https://doi.org/10.1007/s40745-020-00253-

[51] Shaowei Wang, David Lo, Bogdan Vasilescu, and Alexander Serebrenik. 2018. EnTagRec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering* 23, 2 (2018), 800–832.

[52] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[53] Ratnadira Widyasari, Zhipeng Zhao, Thanh Le Cong, Hong Jin Kang, and David Lo. 2023. Topic Recommendation for GitHub Repositories: How Far Can Extreme Multi-Label Learning Go?. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 167–178.

[54] Jiawei Wu, Wenhan Xiong, and William Yang Wang. 2019. Learning to Learn and Predict: A Meta-Learning Approach for Multi-Label Classification. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 4354–4364. https://doi.org/10.18653/v1/D19-1444

[55] Tong Wu, Qingqiu Huang, Ziwei Liu, Yu Wang, and Dahua Lin. 2020. Distribution-balanced loss for multi-label classification in long-tailed datasets. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*. Springer, 162–178.

[56] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. 2013. Tag recommendation in software information sites. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 287–296.

[57] Bowen Xu, Thanh-Dat Nguyen, Thanh Le-Cong, Thong Hoang, Jiakun Liu, Kisub Kim, Chen Gong, Changan Niu, Chenyu Wang, Bach Le, et al. 2023. Are We Ready to Embrace Generative AI for Software Q&A?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1713–1717.

[58] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment analysis for software engineering: How far can pre-trained transformer models go?. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 70–80.

[59] Xin Zhou, Kisub Kim, Bowen Xu, Jiakun Liu, DongGyun Han, and David Lo. 2023. The Devil is in the Tails: How Long-Tailed Code Distributions Impact Large Language Models. *arXiv preprint arXiv:2309.03567* (2023).

[60] Xin Zhou, Bowen Xu, Kisub Kim, DongGyun Han, Thanh Le-Cong, Junda He, Bach Le, and David Lo. 2023. Patchzero: Zero-shot automatic patch correctness assessment. *arXiv preprint arXiv:2303.00202* (2023).

[61] Xin Zhout, Kisub Kim, Bowen Xu, Jiakun Liu, DongGyun Han, and David Lo. 2023. The Devil is in the Tails: How Long-Tailed Code Distributions Impact Large Language Models. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, 40–52.

[62] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*. 19–27.

[63] Liu Zhuang, Lin Wayne, Shi Ya, and Zhao Jun. 2021. A Robustly Optimized BERT Pre-training Approach with Post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*. 1218–1227.