PROJECT 2B: GitHub Repository Topic Classification Based on Textual Metadata

1st Huynh Thai Linh, 2nd Truong Minh Khoa, 3rd Ho Gia Thanh
and Le Nhat Tung
HUTECH University, Vietnam
{huynh.thai.linh2004, truongminhkhoa16, hogiathanh572}@gmail.com and lenhattung@hutech.edu.vn

Abstract

This study compares classical machine learning methods with a Transformer-based model using Parameter-Efficient Fine-Tuning (PEFT) for GitHub Repository Topic Classification Based on Textual Metadata. Four classical models—Logistic Regression, Random Forest, SVC, and KNN—achieved moderate results (F1-scores between 0.56–0.68), showing limited ability to capture semantic context. In contrast, the Mistral-7B model fine-tuned with LoRA and 4-bit quantization achieved strong and consistent performance (precision, recall, F1 $\approx$ 0.95). These findings demonstrate the superior representational capacity of Transformer architectures and confirm that PEFT offers an efficient balance between performance and computational cost for large-scale repository analytics.

Keywords

GitHub repository, README.md, Text Classification, Bert embeddings, natural language processing, Textual Metadata

I. Introduction

In recent years, the rapid growth of *open-source software (OSS)*[1] [1] repositories on platforms such as GitHub has led to an unprecedented abundance of project-related textual data [2]. Each repository contains a variety of metadata and documentation, among which the **README.md** file serves as the primary medium for describing project functionality, purpose, dependencies, and usage [3]. These descriptions, while human-readable, are often unstructured and noisy, posing significant challenges for automated understanding, organization, and retrieval of software projects [4].

To improve discoverability and facilitate downstream analysis, GitHub allows users to assign topics to repositories (e.g., *"machine-learning," "web-development," "data-visualization"*) [5]. However, these topic labels are typically user-generated, inconsistent, and incomplete [6], [7]. Consequently, the ability to automatically classify repositories into topics based on textual metadata has become a crucial research problem that lies at the intersection of *Natural Language Processing (NLP)* and *Software Engineering (SE)* [8].

This study proposes a comprehensive approach for **GitHub repository topic classification based on textual metadata**, focusing primarily on README files as the main semantic source. Using the **GitHub GraphQL API**[2] [9], we collected a large-scale dataset comprising **57,368 README files** spanning over *50 distinct topics*. For each topic, up to 2,000 repositories were crawled across four retrieval criteria — *most starred, most forked, recently updated, and best match* — with randomized page ordering to ensure diversity. After deduplication, approximately *1,000 unique repositories per topic* were retained, supplemented by random or star-range sampling where necessary.

To address the heterogeneity and noisiness of natural language in software documentation, we performed extensive pre-processing and representation learning using **Sentence-BERT embeddings** [10], followed by **parameter-efficient fine-tuning (PEFT)** [11] of large transformer models such as **Mistral-7B** [12] under *low-bit quantization (4-bit)* [13]. This strategy enables high-quality classification performance while maintaining computational efficiency.

The contributions of this paper are threefold:

1. We construct and publicly describe a large-scale, diverse dataset of *GitHub README* files across more than fifty repository topics.
2. We develop a robust text preprocessing pipeline tailored to software documentation, including noise reduction, stopword enrichment with domain-specific tokens, and semantic normalization.
3. We apply and evaluate state-of-the-art transformer-based models for repository topic classification, leveraging *PEFT* and *LoRA* fine-tuning for efficiency.

[1]Open-source software (OSS) repositories provide publicly accessible source code, documentation, and version history for software projects and are commonly hosted on collaborative platforms (e.g., GitHub).

[2]The GitHub GraphQL API was used to crawl README files and metadata from public repositories across multiple topics

The results of this study demonstrate that rich textual metadata from **README** files carries sufficient semantic information to accurately infer repository topics, thereby enabling scalable organization and retrieval of open-source projects.

II. Related Work

The problem of text classification in general, and software repository classification in particular, has attracted significant attention from the research community over the past years. Existing approaches can be broadly categorized into three main groups: traditional machine learning methods, deep learning-based methods, and approaches leveraging large language models built upon the Transformer architecture.

Early approaches primarily relied on bag-of-words or TF-IDF representations extracted from README or source code comments, combined with classical machine learning algorithms such as Support Vector Machines (SVM), Naïve Bayes, or Random Forests. For instance, studies by Pan et al. (2017) [14] Di Sipio et al. (2020) [15] or Xin & Xia et al. (2017) [16] leveraged TF-IDF representations of README contents to classify or recommend repositories categories, achieving reasonable performance but limited generalization due to lexical sparsity and lack of semantic understanding.

Recent advances in representation learning have significantly improved the quality of textual features for repository understanding. Techniques such as Doc2Vec, word embeddings, and more recently, transformer-based embeddings (e.g., BERT, RoBERTa, Sentence-BERT) capture contextual semantics beyond surface-level token frequencies [17]. Works like Zhang et al. (2021) [18] demonstrated that BERT embeddings effectively model repository descriptions, outperforming TF-IDF in topic and intent classification tasks.

Repository mining studies have also leveraged multimodal metadata — combining README text, project descriptions, dependency files, and commit messages — to enrich semantic representation. Nonetheless, README files remain the most reliable and consistently available textual source, making them an ideal focus for large-scale topic classification.

Parameter-efficient fine-tuning (PEFT) methods, such as LoRA and QLoRA, have recently enabled adapting large language models (LLMs) like Mistral or LLaMA to specialized tasks with minimal computational cost. These approaches have shown strong performance in text classification under resource-constrained environments. To our knowledge, few studies have applied such techniques to software repository classification at scale. [19]

Therefore, this work extends existing literature by integrating semantic-rich embeddings and PEFT-based fine-tuning for efficient, scalable, and accurate classification of GitHub repositories based solely on textual metadata.

III. Methodology

The research methodology is systematically designed and consists of the following key stages: data collection and dataset construction; text preprocessing and normalization; text representation and feature extraction; model building and training; and model performance evaluation. An overview of the entire process is illustrated in Figure 1.

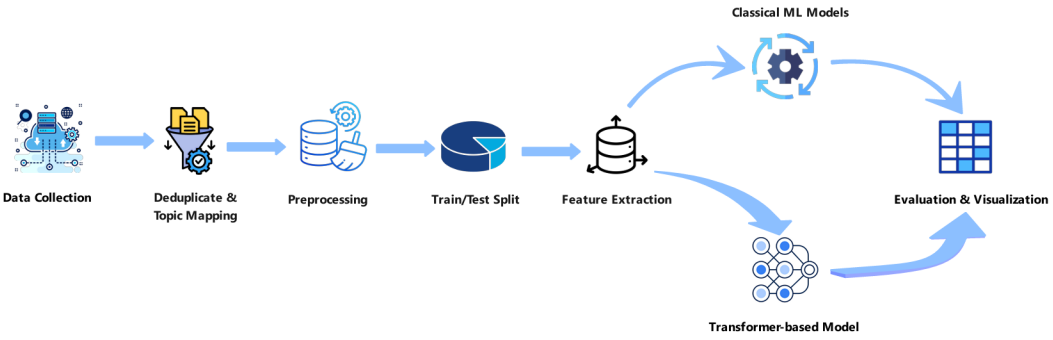

Figure 1: Pipeline for Feature Extraction and Classification of GitHub README Files.

A. Data Collection and Taxonomy Construction

To construct a comprehensive and representative dataset for GitHub repository topic classification, we collected textual data from GitHub using its GraphQL API. The resulting dataset comprises **57,368 README.md files** from repositories covering more than *50 distinct topics* within the domain of *Information Technology (IT)*.

- **Topic selection:** We began by identifying over 50 GitHub topics that are both popular and domain-specific within IT *(e.g., data science, web development, programming languages, software tools, etc.)*.
- **Repository crawling:** For each topic, we systematically retrieved up to *2,000 repositories* based on four complementary ranking dimensions: (1) *Most Starred* — repositories receiving the highest community attention; (2) *Most Forked* —

repositories with the greatest collaborative engagement; (3) *Recently Updated* — repositories with ongoing and active development; and (4) *Best Match* — randomly sampled repositories to increase topical diversity. Each dimension contributed approximately 500 repositories, with randomized page ordering applied to minimize sampling bias.

- **Deduplication and normalization:** After aggregation, all repositories were deduplicated based on their unique repository IDs, retaining up to *1,000 unique repositories per topic*. In cases where the target quota was not met, additional repositories were gathered through random pagination or star-range filtering to preserve balance across topics.
- **Metadata extraction:** For each repository, essential metadata such as repository ID, name, description, topics, and programming language were extracted and stored in a structured CSV format. The corresponding README.md content was separately crawled and stored in JSONL format for subsequent text analysis.

To facilitate higher-level semantic organization, the *50 fine-grained GitHub topics* were mapped into *10 broader categories*, forming a hierarchical taxonomy that represents major domains within IT. These categories later serve as the target labels for the multi-class classification task.

In total, the final dataset contains **57,368 README.md** files and their associated metadata, providing a large-scale and diverse corpus for subsequent preprocessing, feature extraction, and *classification analysis.*

Table I: Description of the collected datasets

File	Column	Description	Type	Non-Null Count
`github_repos.csv`	repo_id	Unique repository identifier	object	57,383
	name	Repository name	object	57,382
	full_name	Full repository path (user/name)	object	57,383
	description	Textual description of the repository	object	56,274
	topics	List of associated GitHub topics	object	57,383
	language	Main programming language	object	52,730
	stars_count	Number of stars	int64	57,383
	forks_count	Number of forks	int64	57,383
	created_at	Repository creation date	object	57,383
	updated_at	Last updated date	object	57,383
	url	Repository URL	object	57,383
	category	Target topic label for classification	object	57,383
`readme_data.json`	repo_id	Repository identifier	object	57,383
	full_name	Repository full name	object	57,383
	readme	Extracted README content	object	57,383
	timestamp	Retrieval timestamp	datetime64[ns]	57,383

B. Data Preprocessing

After compiling the dataset of **57,383 README.md files**, Each README document was preprocessed using a custom text normalization pipeline designed to remove noise typical in software documentation. The preprocessing steps included:

1. Code and URL removal – Markdown code blocks, inline code segments, and hyperlinks were stripped using regular expressions.

2. Markdown syntax cleaning – Header marks, bold, and italic formatting were normalized to plain text.

3. Tokenization and lemmatization – Words were tokenized using NLTK's *word_tokenize* function and lemmatized via *WordNetLemmatizer* to reduce inflectional forms.

4. Stopword filtering – Standard English stopwords were extended with a domain-specific list of programming-related terms *(e.g., install, build, repository, command, example, file,...)* to minimize non-informative content.

5. Lowercasing and noise removal – All text was lowercased and cleaned from punctuation, digits, and special characters.

This process yielded a cleaned corpus in which each README file was represented by its semantically normalized text, suitable for both embedding-based and transformer-based modeling.

C. Feature Representation

After preprocessing, the textual data must be transformed into numerical vector representations to be processed by machine learning models. Our approach follows two main directions:

- **Sentence Embeddings for traditional models:** The **all-MiniLM-L6-v2** model from the *SentenceTransformer* library was employed. This model converts each preprocessed README.md file into a dense feature vector of *384 dimensions.* This approach effectively captures the semantic meaning of the entire text.
  This embedding captures semantic information of the text. The full corpus is represented as:

$$X = [x_1, x_2, \ldots, x_N]^T, \quad y = [y_1, y_2, \ldots, y_N]$$

- **Transformer-based Tokenization:** For the large language model, we utilized the *AutoTokenizer* corresponding to the **Mistral-7B-v0.1** base model. Each text was tokenized into a sequence of token IDs, with padding and truncation techniques applied to ensure that all input sequences have a fixed length of *512 tokens.*

D. Model Architectures

(1) Classical Machine Learning Models: We benchmarked several classical classifiers trained on the MiniLM embeddings:

- **Logistic Regression (LR):** A multinomial logistic regression model designed for multi-class classification tasks.

$$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b), \quad \sigma(z) = \frac{1}{1+e^{-z}} \tag{1}$$

where w and b are model parameters optimized by minimizing cross-entropy loss:

$$\text{LCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

- **Random Forest (RF):** An ensemble learning model based on a collection of decision trees.
An ensemble of T decision trees $\{h_t\}_{t=1}^{T}$ producing majority-vote predictions:

$$\hat{y} = \text{mode}\{h_t(x)\}_{t=1}^{T} \tag{2}$$

- **Support Vector Machine (SVM):** A model that seeks an optimal hyperplane to separate data classes. Using a linear kernel and the soft-margin technique, the optimization problem of SVC can be formulated as follows:

$$\min_{\mathbf{w},\,b}\ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0,\,1-y_i(\mathbf{w}^T\mathbf{x}_i+b)\right) \tag{3}$$

where C controls the margin–error tradeoff.

- **K-Nearest Neighbors (KNN):** classification model that assigns labels based on the similarity to the nearest neighboring instances.

$$\hat{y} = \text{mode}\{\, y_i : \mathbf{x}_i \in KNN(\mathbf{x})\,\} \tag{4}$$

All models were trained using class-weight balancing to mitigate label imbalance across topic categories.

(2) Transformer-based Fine-Tuning (Mistral-7B + LoRA):

To leverage contextual understanding, we fine-tuned the **Mistral-7B** model using **Parameter-Efficient Fine-Tuning (PEFT)** via **Low-Rank Adaptation (LoRA)**.

In LoRA, instead of updating the full weight matrix $\mathbf{W}_0 \in \mathbb{R}^{d\times k}$, a low-rank update is introduced:

$$\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{BA} \tag{5}$$

where:

- $\mathbf{A} \in \mathbb{R}^{r\times k}$
- $\mathbf{B} \in \mathbb{R}^{d\times r}$
- $r \ll \min(d,k)$ is the low-rank dimension (in this work, $r = 16$)

This approach drastically reduces trainable parameters while maintaining model expressiveness.

The training objective minimizes the **cross-entropy loss** over all tokenized samples:

$$\text{L}_{\text{PEFT}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{|C|} y_{ic}\log(p_{ic}) \tag{6}$$

where $p_{ic} = \text{softmax}(\mathbf{z}_i)_c$ is the predicted probability for class c.

E. Experimental Setup and Evaluation Metrics

The dataset was divided into a *training set (80%)* and a *test set (20%)* using the stratified sampling technique.

Table II: Train-test split (80:20) of the dataset

Dataset	Number of samples	Number of features
Train	45,894	4
Test	11,474	4
Total	57,368	4

The performance of the classification models was evaluated using standard metrics: *precision (P), recall (R), F1-score (F1)*, and *accuracy (A)*. These metrics are defined as:

$$\text{Precision } (P) = \frac{TP}{TP+FP}, \quad \text{Recall } (R) = \frac{TP}{TP+FN}, \quad F_1 = \frac{2 \cdot P \cdot R}{P+R}, \quad \text{Accuracy } (A) = \frac{TP+TN}{TP+TN+FP+FN} \tag{7}$$

where:

- TP (True Positives): the number of samples correctly predicted as belonging to a given class.
- FP (False Positives): the number of samples incorrectly predicted as belonging to the class.
- FN (False Negatives): the number of samples belonging to the class but predicted as another class.
- TN (True Negatives): the number of samples correctly predicted as not belonging to the class.
- **Precision** measures the proportion of correctly predicted positive instances among all instances predicted as positive, reflecting the model's exactness.
- **Recall** measures the proportion of correctly predicted positive instances among all actual positive instances, reflecting the model's ability to capture relevant cases.
- **F1-score** is the harmonic mean of precision and recall, providing a balanced measure when both false positives and false negatives are important.
- **Accuracy** represents the overall proportion of correctly classified instances among all instances.

Visualization of performance metrics across the 10 taxonomy classes was conducted using heatmaps of classification reports and confusion matrices, enabling qualitative analysis of topic misclassification.

This section outlines the experimental configuration, parameter settings, and evaluation methodology to ensure reproducibility and reliable comparison across classification techniques.

IV. Baselines / Comparison Methods

To provide a comprehensive evaluation of our proposed methodology, we established several baseline models that represent widely-used approaches in textual metadata classification.

A. Classical Machine Learning Baselines

We first benchmarked traditional classifiers trained on MiniLM sentence embeddings of README files. The selected models include:

- Logistic Regression (LR)
- Support Vector Machine (SVM) with linear kernel
- Random Forest (RF)
- K-Nearest Neighbors (KNN)

All models were trained with balanced class weights to account for potential label imbalance.

B. Transformer-Based Baselines

To assess the impact of contextualized embeddings and fine-tuning, we included two transformer-based baselines:

- **Full Fine-Tuning of Mistral-7B:** The entire model is updated on the training set to learn task-specific representations.
- **LoRA Parameter-Efficient Fine-Tuning:** Only low-rank adapter layers are trained while freezing the original model weights, significantly reducing computational and memory costs while retaining high performance.

C. Evaluation

All baselines were evaluated on the same test set using *accuracy, precision, recall, and F1-score*, both at the class level and globally. This setup enables a fair comparison between traditional machine learning models and modern transformer-based approaches, highlighting the trade-offs between computational efficiency and classification performance.

V. Experimental Results

This section presents a comprehensive evaluation of our GitHub repository topic classification framework. We begin by describing the dataset characteristics and preprocessing steps, followed by the results of classical machine learning baselines and transformer-based models using PEFT techniques. All experiments were executed on a workstation equipped with an NVIDIA A6000 GPU and 220 GB RAM.

A. Dataset Overview

We constructed a large-scale dataset of GitHub repositories by crawling *57,368 README files* across 50 topics, targeting *2,000 repositories per topic* using four sampling strategies: *500 most starred, 500 most forked, 500 recently updated, and 500 best match* (with randomized page orders). Deduplication was applied to select *unique repositories per topic*, and if a topic had fewer than 1,000 repositories, additional sampling based on star range was performed to reach. Finally, the 50 topics were grouped into *10 higher-level categories* representing distinct domains within information technology.

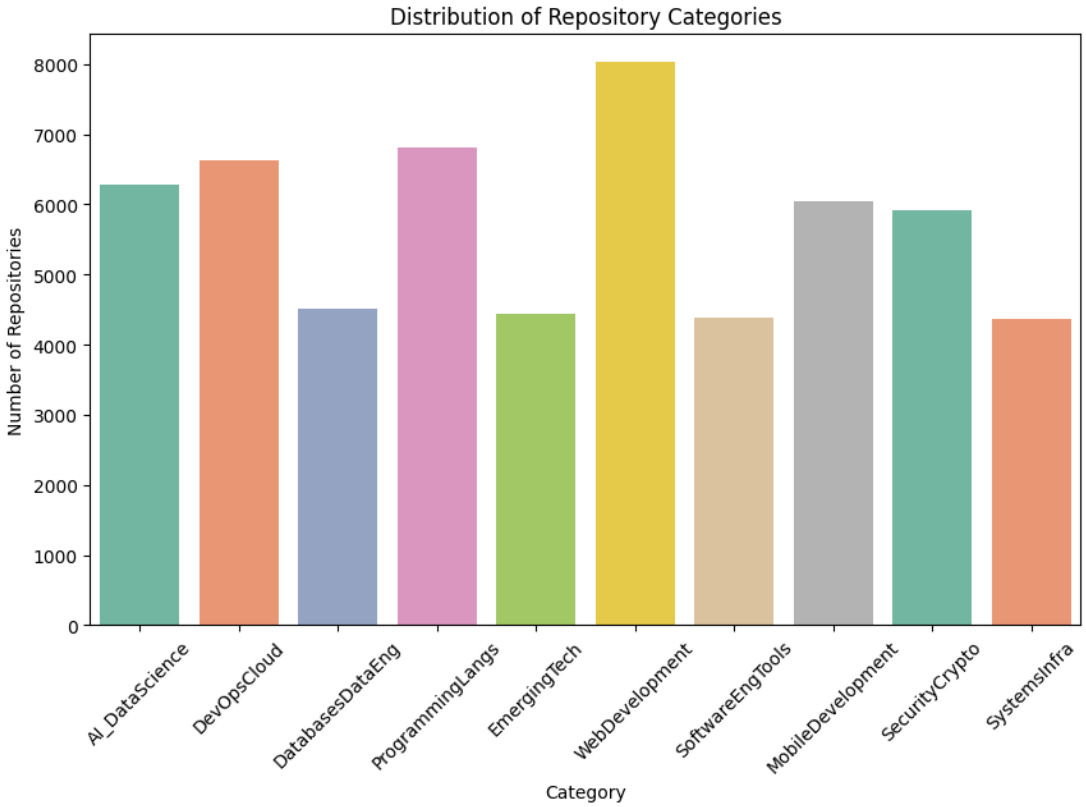

Figure 2: Distribution of Repository Categories

The visualization depicts the distribution of repository categories extracted from the GitHub dataset, revealing distinct patterns in how projects are concentrated across technological domains. Web Development emerges as the most represented category, reaching around 8,000 repositories. This reflects the dominance of web-related frameworks, front-end and back-end technologies, and the continuous evolution of digital service development on GitHub.

The categories Programming Languages, DevOps/Cloud, and AI/Data Science also maintain considerable presence, each exceeding 6,000 repositories. These areas embody the backbone of modern software ecosystems—combining core language experimentation, deployment automation, and intelligent computation. The visible balance among these fields indicates a healthy interaction between foundational and emerging aspects of programming communities.

Meanwhile, Mobile Development and Security/Crypto stand at intermediate levels, emphasizing the importance of secure mobile applications and decentralized systems in current development trends. In contrast, Databases/Data Engineering, Software Engineering Tools, Emerging Tech, and Systems/Infra are represented at lower frequencies, approximately between 4,400 and 4,600 repositories. These categories tend to be more specialized, often requiring advanced technical expertise and serving as enablers for the higher-level domains mentioned earlier.

This variation in repository counts across categories provides a rich context for the classification task, highlighting the heterogeneous nature of textual metadata within open-source ecosystems.

The dataset was split into training and testing sets using stratified sampling, with *80% for training* and *20% for evaluation*, ensuring balanced class representation.

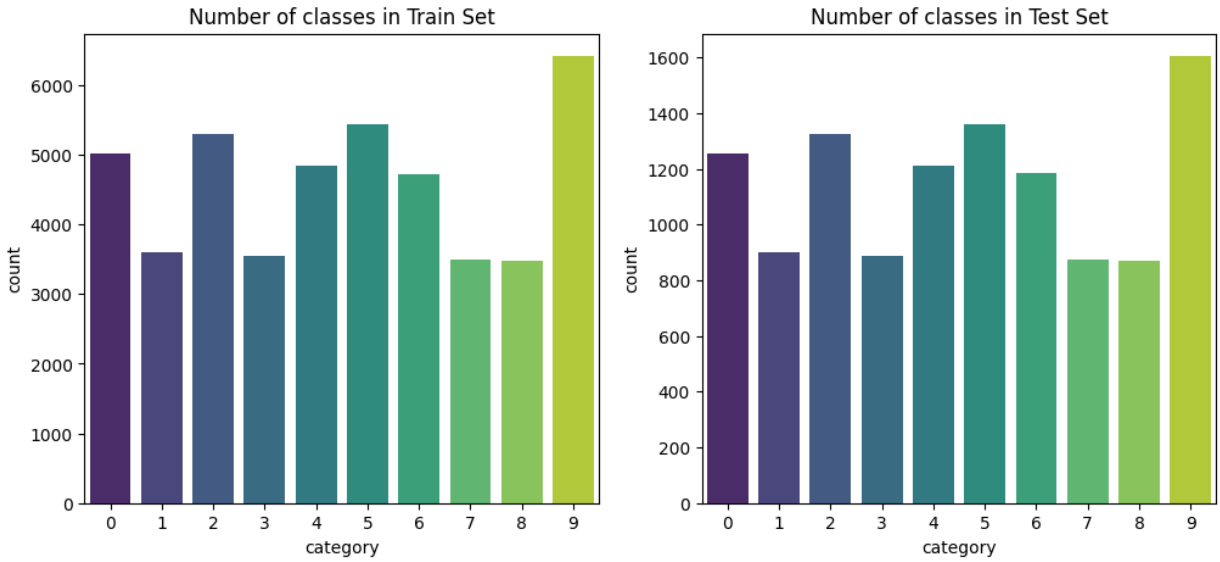

Figure 3: Class distribution in training and testing sets

The figure presents the distribution of repository categories in the training and testing datasets, following an 80:20 split ratio. This proportional division ensures that both subsets retain the original diversity of topics while allowing sufficient data for model learning and evaluation.

In the training set (left chart), category 9 contains the largest number of samples, exceeding 6,000 repositories, while categories 1, 3, 7, and 8 have smaller counts, approximately around 3,400–3,600. Other categories, such as 0, 2, 4, 5, and 6, maintain medium levels, ranging between 4,700 and 5,500 repositories. This distribution suggests a relatively mild imbalance that is unlikely to distort the model's learning process.

The test set (right chart) mirrors this structure, with each category's count being roughly 20% of its training counterpart. The proportional alignment between both subsets confirms that the data splitting preserved the statistical characteristics of the entire dataset. Consequently, this balanced 80:20 partition enhances the fairness of performance evaluation and ensures that the classifier can generalize effectively across diverse repository topics.

B. Classical Machine Learning Models

We evaluated four traditional classifiers on sentence embeddings generated using *all-MiniLM-L6-v2*:

- Logistic Regression
- Random Forest Classifier
- Support Vector Classifier (linear kernel)
- k-Nearest Neighbors (k-NN)

All models were trained using *balanced class weights* to mitigate category imbalance. Performance was evaluated on the test set using *precision, recall, F1-score,* and *accuracy.*

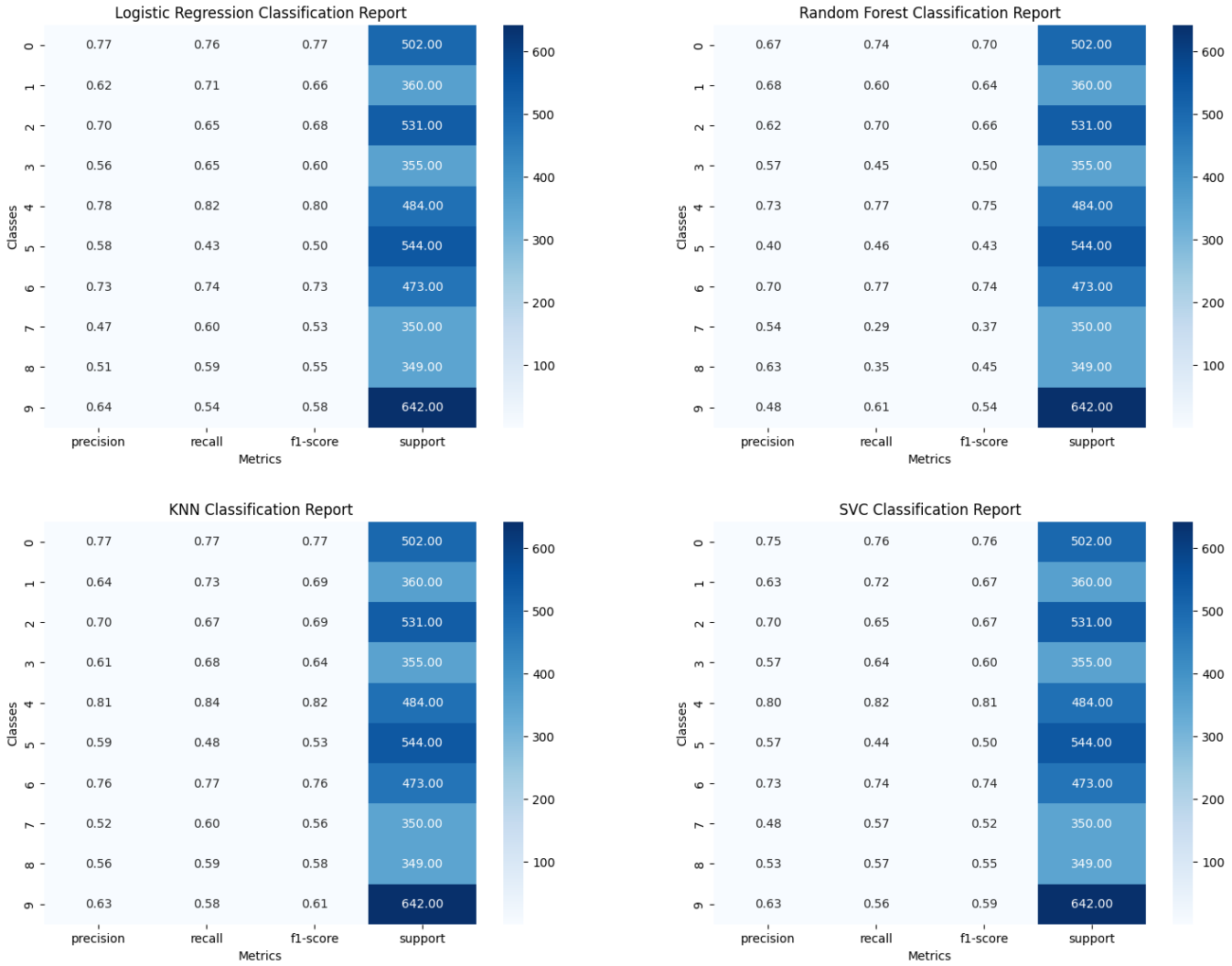

Figure 4: Classification report of the models.

Through comparative evaluation of Logistic Regression, Random Forest, Support Vector Classifier (SVC), and K-Nearest Neighbors (KNN), we observe distinct strengths and weaknesses in how each model handles the textual metadata of GitHub repositories.

The Logistic Regression model demonstrates stable and well-balanced results across most classes, maintaining consistent F1-scores around 0.70–0.80. It captures dominant topics effectively, particularly in structured or high-frequency classes (e.g., classes 0, 4, and 6), though it struggles with underrepresented ones like class 5.

In contrast, Random Forest exhibits greater variance between classes, suggesting sensitivity to class imbalance. While it performs well for a few classes such as 4 and 6, it suffers from performance drops in minority categories (e.g., 7 and 8), possibly due to overfitting to frequent textual patterns.

The SVC model offers a balanced trade-off between bias and variance. Its kernel-based structure enhances boundary separation for complex textual data, achieving high F1-scores for classes 4 and 6 (0.81 and 0.74 respectively). SVC shows robustness in handling both frequent and moderate classes, though its efficiency decreases for ambiguous or overlapping topics.

Lastly, KNN provides competitive results, particularly for classes where local textual similarity dominates (e.g., 0, 4, 6). However, the model's reliance on neighborhood density makes it less effective in sparse or high-dimensional contexts, leading to modest results for classes like 5 and 8.

From a holistic perspective, SVC and Logistic Regression emerge as the most stable models, maintaining consistent precision–recall balance and superior generalization. Random Forest and KNN, while effective for certain classes, exhibit larger fluctuations, indicating the need for further feature normalization or dimensionality reduction.

Table III: Comparison of Classification Metrics Across Four Models

Model	Precision	Recall	F1-Score
Logistic Regression	0.66	0.69	0.66
Random Forest	0.58	0.62	0.56
SVC	0.66	0.69	0.67
KNN	0.67	0.70	0.68

The table summarizes the average performance of the four models across all classes. KNN and SVC exhibit the highest F1-scores (0.68 and 0.67), indicating effective balance between precision and recall. Logistic Regression performs comparably well, while Random Forest lags slightly behind due to class imbalance sensitivity.

C. Transformer-based Classification with PEFT

We further fine-tuned the *Mistral-7B* model using *LoRA (Low-Rank Adaptation)* and *4-bit quantization* for efficient training on the large dataset.

- Tokenization was applied with *maximum sequence length 512.*
- Data was collated using **DataCollatorWithPadding** for optimized batching.
- Training was performed for 1 epoch with batch size 256, gradient accumulation of 7 steps, and mixed-precision *(fp16)* computation.

Evaluation: Predictions on the test set were obtained, and class probabilities were computed using softmax. Performance was summarized using precision, recall, F1-score, and accuracy.

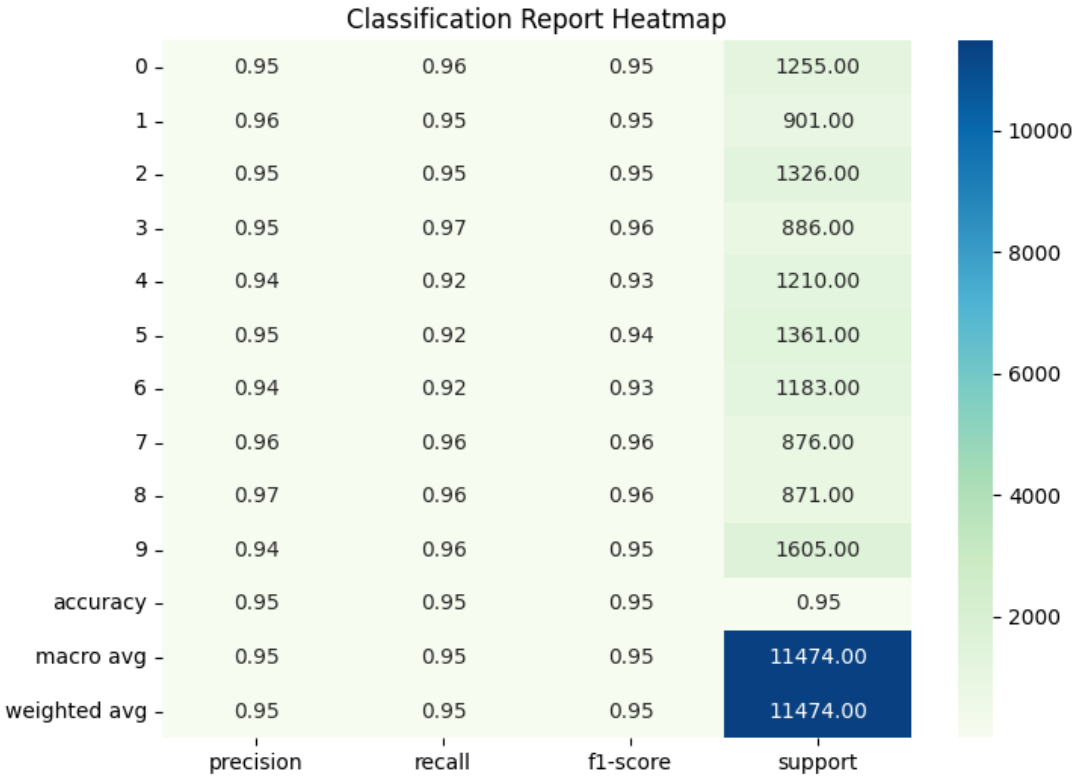

Figure 5: Classification Report of fine-tune

The heatmap above presents the classification report of the *Mistral-7B* transformer-based model fine-tuned with *LoRA (Low-Rank Adaptation)* and *4-bit quantization under a PEFT (Parameter-Efficient Fine-Tuning)* setting. This model was trained on a large GitHub dataset for the task of Repository Topic Classification Based on Textual Metadata.

From the visualization, we observe consistent high performance across all classes, with *precision, recall* and *F1-score* values around **0.94–0.97**. The overall accuracy reaches **0.95**, reflecting strong generalization and balanced learning.

- Classes such as *8* and *3* achieve the highest precision and recall (≈ 0.97 and 0.96–0.97), showing the model's effectiveness in identifying these repository topics.

- Conversely, classes *4, 5*, and *6* exhibit slightly lower recall values ≈ 0.92 , implying occasional under-detection for these categories—possibly due to semantic overlap or data imbalance in those classes.
- The macro average and weighted average metrics both equal 0.95, confirming uniform performance across categories, without overfitting to high-frequency classes.

The support distribution indicates the dataset is moderately balanced, with each class containing between ≈ 870 *and 1600* samples, totaling *11,474 instances*. This contributes to stable performance across the entire label space.

In summary, the *PEFT-based fine-tuning of Mistral-7B* demonstrates high accuracy and efficiency, validating the effectiveness of *LoRA and quantized* computation for large-scale textual classification tasks on GitHub metadata.

Table IV: Performance metrics of the classification models

Metric Type	Precision	Recall	F1-score	Accuracy
Per-class range	0.94–0.97	0.92–0.97	0.93–0.96	–
Macro Avg	0.95	0.95	0.95	–
Weighted Avg	0.95	0.95	0.95	0.95

The model achieves uniform high performance with 0.95 overall accuracy, confirming the robustness of transformer-based fine-tuning with LoRA under PEFT for GitHub topic classification.

D. Summary of Results

Table V: Classification Metrics for 4 Classical ML Models and Transformer-based PEFT

Model / Metric Type	Precision	Recall	F1-Score	Accuracy
4 Classical Machine Learning Models				
Logistic Regression	0.66	0.69	0.66	–
Random Forest	0.58	0.62	0.56	–
SVC	0.66	0.69	0.67	–
KNN	0.67	0.70	0.68	–
Transformer-based Classification with PEFT				
Per-class range	0.94–0.97	0.92–0.97	0.93–0.96	–
Macro Avg	0.95	0.95	0.95	–
Weighted Avg	0.95	0.95	0.95	0.95

The comparison results in Table V reveal a clear performance gap between classical machine learning algorithms and the Transformer-based PEFT model. The traditional models — Logistic Regression, Random Forest, SVC, and KNN — achieved relatively modest results, with F1-scores fluctuating between 0.56 and 0.68. Among them, KNN slightly outperformed others, reaching an F1-score of 0.68, while Random Forest showed the weakest result at 0.56. These outcomes suggest that shallow models, relying primarily on linear or distance-based decision boundaries, struggle to capture the complex semantic relations embedded in textual metadata from GitHub repositories.

By contrast, the Transformer-based model fine-tuned with LoRA under the PEFT framework demonstrated a remarkable leap in all metrics, achieving around 0.95 for precision, recall, and F1-score. This reflects the capacity of large-scale transformer architectures such as Mistral-7B to model long-range dependencies and nuanced contextual representations. The consistent performance across all classes (0.94–0.97) indicates strong robustness, even when the dataset contains heterogeneous and overlapping topic labels. Moreover, the use of 4-bit quantization and LoRA adaptation ensures computational efficiency without compromising classification accuracy.

These findings highlight a fundamental shift in performance potential between conventional feature-based methods and modern language model-based architectures. While classical models provide lightweight, interpretable baselines, the Transformer-based PEFT model demonstrates deep contextual understanding and superior generalization — a crucial advantage for large-scale, text-driven classification tasks such as GitHub topic prediction.

VI. Discussion

The experimental findings demonstrate a clear distinction between the representational capabilities of classical machine learning models and the Transformer-based PEFT approach. Traditional classifiers, including Logistic Regression, Random Forest, SVC, and KNN, exhibit limited ability to generalize semantic and contextual information from GitHub repository metadata. Their reliance on static embeddings and linear or distance-based separation boundaries results in moderate performance, with F1-scores ranging from 0.56 to 0.68. This outcome suggests that although sentence embeddings provide meaningful numerical representations, classical models are inherently constrained in modeling hierarchical relations and subtle contextual variations across repository topics.

In contrast, the Mistral-7B Transformer fine-tuned via LoRA under the PEFT configuration achieved uniform and substantially higher performance across all evaluation metrics (precision, recall, and F1 ≈ 0.95). The model's ability to handle contextual dependencies through self-attention and to adapt efficiently through low-rank updates explains this performance leap. Furthermore, its consistent per-class results (0.94–0.97) highlight robustness against topic imbalance and semantic overlap. The integration of 4-bit quantization also underscores the practicality of deploying large-scale models with limited computational resources, balancing both efficiency and accuracy. These findings collectively validate that parameter-efficient fine-tuning is a viable and scalable approach for real-world repository classification tasks.

From a methodological standpoint, the results also imply that the combination of pretrained language models and PEFT techniques bridges the gap between performance and resource optimization. While fully fine-tuning large models is computationally expensive, PEFT effectively captures domain-specific nuances with a fraction of the trainable parameters. Thus, this approach provides a sustainable pathway for continuous improvement in software repository analytics and other large-scale NLP applications where interpretability and efficiency are both essential.

VII. Conclusion

This study presented a comprehensive comparison between classical machine learning methods and a Transformer-based PEFT framework for the task of GitHub repository topic classification using textual metadata. The classical models established a reasonable baseline, achieving F1-scores up to 0.68, but their limited representational depth hindered performance on semantically complex data. In contrast, the Mistral-7B model fine-tuned with LoRA and quantized to 4-bit precision achieved a consistent accuracy of 0.95, demonstrating not only superior contextual understanding but also the feasibility of efficient large-scale adaptation.

The results confirm that parameter-efficient fine-tuning enables high-capacity transformers to be applied effectively in domain-specific classification problems without the need for full retraining. This approach maintains state-of-the-art performance while significantly reducing computational costs, making it highly suitable for scalable deployment in software engineering and open-source analytics. Future work could explore integrating repository-level graph features, multimodal signals (e.g., code snippets, issue comments), or task-specific prompt adaptation to further enhance the model's interpretability and generalization across unseen repository domains.

REFERENCES

[1] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. German, "Open source-style collaborative development practices in commercial projects using github," 05 2015.

[2] G. Staff. Octoverse: AI leads python to top language as the number of global developers surges. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-2024/

[3] T. Kinsman, M. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use github actions to automate their workflows?" 03 2021.

[4] S. Gupta and S. Gupta, "Natural language processing in mining unstructured data from software repositories: a review," *Sādhanā*, vol. 44, p. 244, 11 2019.

[5] Classifying your repository with topics. [Online]. Available: https://docs-internal.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/classifying-your-repository-with-topics

[6] P. Nguyen, J. Rocco, C. Di Sipio, M. Shakya, D. Di Ruscio, and M. Di Penta, "Automatic categorization of github actions with transformers and few-shot learning," 07 2024.

[7] M. M. Aslam, M. Farhan, A. Raza, J. Iqbal, and M. Iqbal, "A smart model for categorization of github repositories," *KIET Journal of Computing and Information Sciences*, vol. 6, p. 50, 07 2024.

[8] L. Zhao, W. Alhoshan, A. Ferrari, K. Letsholo, M. Ajagbe, R. Batista-Navarro, and E.-V. Chioasca, "Natural language processing (nlp) for requirements engineering: A systematic mapping study," vol. https://arxiv.org/abs/2004.01099, 12 2020.

[9] GitHub GraphQL API documentation. [Online]. Available: https://docs-internal.github.com/en/graphql

[10] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 01 2019, pp. 3973–3983.

[11] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 06 2021.

[12] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[13] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: efficient finetuning of quantized llms," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.

[14] M. Martinc, I. Škrjanec, K. Zupan, and S. Pollak, "Pan 2017: Author profiling-gender and language variety prediction (notebook for pan at clef 2017, 2nd place)," 02 2018.

[15] A. Dahou and B. Mathiak, "Subject classification of software repository," 01 2023, pp. 30–38.

[16] W. Xu, X. Sun, X. Xia, and X. Chen, "Scalable relevant project recommendation on github," 09 2017, pp. 1–10.

[17] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," 2014. [Online]. Available: https://arxiv.org/abs/1405.4053

[18] N. M. Gardazi, A. Daud, M. K. Malik, A. Bukhari, T. Alsahfi, and B. Alshemaimri, "BERT applications in natural language processing: a review," vol. 58, no. 6, p. 166. [Online]. Available: https://doi.org/10.1007/s10462-025-11162-5

[19] Z. Liu, J. Lyn, W. Zhu, and X. Tian, "Alora: Allocating low-rank adaptation for fine-tuning large language models," 01 2024, pp. 622–641.