Project-Specific Code Summarization with Meta-Learning and Explainability Techniques

Quang-Huy Nguyen[1,2], Hoai-Phong Le[1,2] and Bac Le[1,2]

[1] Faculty of Information Technology, University of Science,
Ho Chi Minh City, Vietnam

[2] Vietnam National University, Ho Chi Minh City, Vietnam

Abstract. Code summarization generates natural language descriptions for code snippets, enhancing readability and maintainability. While current methods perform well with large-scale datasets, they struggle in low-resource scenarios typical of smaller and newer projects. Additionally, developers need summaries that capture project-specific characteristics rather than generic descriptions. To address these challenges, we propose a meta-learning-based training framework that adapts the model to individual projects as distinct tasks, even with minimal data. We introduce a strategy for selecting support projects to boost the framework's effectiveness. Experiments on eight real-world projects show that our method outperforms the baseline approach. Furthermore, we use explainability techniques to clarify the prediction process and identify potential issues.

1 Introduction

In software development, programmers spend up to 58% of their time comprehending source code [1], and natural language summaries can significantly reduce this effort. However, summarizing manually is often time-consuming and labor-intensive. This emphasizes the need for automatic code summarization, a task that generates a concise description that captures a code snippet's functionality.

Researchers have explored various approaches, from traditional rule-based [2] and retrieval-based [3] methods to modern deep learning techniques [4]. However, current methods still face significant challenges, particularly in adapting to projects with diverse coding conventions and structures. Limited labeled data further exacerbates this issue, restricting model performance in new or low-resource projects. Furthermore, the "black-box" nature of deep learning models hampers transparency and interpretability, limiting trust in their predictions.

To address these challenges, we propose a training framework based on Model-Agnostic Meta-Learning (MAML) [5] for adaptable code summarization. Our approach treats each project as a distinct task, enabling the model to adapt to new projects with minimal data by leveraging a selected set of support projects to enhance MAML's effectiveness. Additionally, we analyze attention matrices to visualize the model's focus areas within the code and conduct causal experiments to identify potential issues within the model.

Our contributions are as follows: (1) We propose a training framework for code summarization that tackles data limitations while enabling project-specific

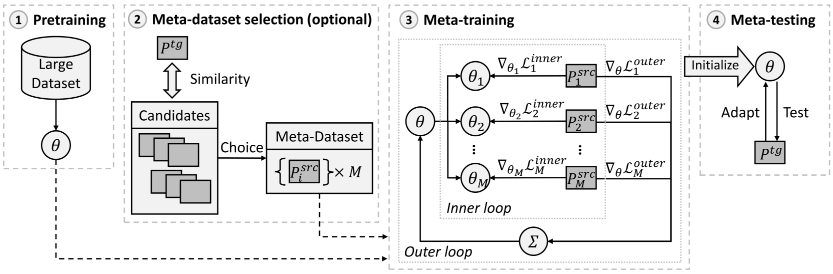
1 Pretraining
Large Dataset
θ
2 Meta-dataset selection (optional)
P^{tg}
Similarity
Candidates
Choice
Meta-Dataset
$\{P_i^{src}\} \times M$
3 Meta-training
θ
θ_1
θ_2
θ_M
$\nabla_{\theta_1}\mathcal{L}_1^{inner}$
$\nabla_{\theta_2}\mathcal{L}_2^{inner}$
$\nabla_{\theta_M}\mathcal{L}_M^{inner}$
P_1^{src}
P_2^{src}
P_M^{src}
$\nabla_{\theta}\mathcal{L}_1^{outer}$
$\nabla_{\theta}\mathcal{L}_2^{outer}$
$\nabla_{\theta}\mathcal{L}_M^{outer}$
Inner loop
Σ
Outer loop
4 Meta-testing
Initialize
θ
Adapt
Test
P^{tg}

Fig. 1: The overall framework for project-specific code summarization.

adaptation; (2) We introduce a meta-dataset selection strategy to optimize training for target projects; (3) Extensive experiments on real-world projects validate our approach and assess factors affecting performance; (4) We apply explainability techniques to clarify the model's predictions and uncover potential issues.

2 Related work

The field of automatic code summarization has evolved significantly over the past decade. Early rule-based methods like [2] relied on handcrafted rules but struggled with generalization. Information extraction techniques like [3], improved on this by extracting relevant data from existing sources, though they mainly captured lexical information, neglecting structural aspects of code. In recent years, deep learning models like CODE-NN [4], GNN [6], and Transformers [7] have transformed the field by learning both semantic and structural features of code. Some models incorporating Abstract Syntax Trees (ASTs) like Hybrid-DeepCom [8] have significantly improved performance and generalization.

MAML [5] have shown effectiveness in adapting to new tasks with minimal data in several NLP applications such as text summarization [9], machine translation [10]. However, their lack of interpretability poses challenges in identifying potential issues. Our approach builds on these foundations, integrating meta-learning to enhance adaptability for project-specific tasks while also incorporating techniques to improve explainability in code summarization.

3 Our appproach

3.1 Project-specific training framework with limited data

We utilized Hybrid-DeepCom [8] as the base model, which extends the vanilla Seq2Seq architecture by incorporating structural information from AST to enhance performance. Instead of using GRU in the original model, we employed Bi-GRU to better capture bidirectional dependencies in the code. Inspired by [11], we introduce a training framework that leverages the MAML algorithm, enabling the model to adapt effectively to target projects with limited training data. Figure 1 provides an overview of the proposed framework, which consists of four key steps detailed below.

Pre-training. We pre-train the model on a large-scale dataset before applying MAML. This reduces the risk of overfitting, particularly when each task's training data is limited [9].

Meta-dataset selection. A meta-dataset is a set of multiple projects used to train the model with meta-learning. Selecting a suitable meta-dataset is essential, as using all available projects is impractical. The meta-dataset should consist of projects most similar to the target project to ensure strong generalization. We use the cosine similarity between semantic representations of projects. Each code snippet is encoded into hidden state vectors via the base model's pre-trained code encoder. For each project, the representations of all its snippets are averaged. Following [12], we apply global max pooling over the vector sequence to generate its semantic representation R_P. Finally, cosine similarity between the target project P^{tg} a candidate project P_i^c is computed as:

$$\text{cosine}(R_{P^{tg}}, R_{P_i^c}) = \frac{R_{P^{tg}} \cdot R_{P_i^c}}{||R_{P^{tg}}|| \cdot ||R_{P_i^c}||} \tag{1}$$

The top-ranked projects based on this similarity are selected for the meta-dataset. We will discuss the effectiveness of this selection in Section 4.2.

Meta-training. MAML operates with two optimization loops. Specifically, each project P_i^{src} in the meta-dataset is divided into a support set $\mathcal{D}_i^{\text{sup}}$ and a query set $\mathcal{D}_i^{\text{qry}}$. In the inner loop, copied parameters θ_i are updated by adapting to the task P_i using the support set $\mathcal{D}_i^{\text{sup}}$ through gradient descent for the number of steps as Equation 2. For each θ_i, we calculate the losses on the query set $\mathcal{D}_i^{\text{qry}}$. In the outer loop, a meta-update is performed on the original model parameters θ by aggregating query losses across source tasks as Equation 3.

$$\theta_i = \theta_i - \alpha \nabla_{\theta_i} \mathcal{L}_i^{\text{inner}}(\theta_i, \mathcal{D}_i^{\text{sup}}), \tag{2}$$

$$\theta = \theta - \beta \nabla_{\theta} \sum_{i=1}^{M} \mathcal{L}_i^{\text{outer}}(\theta_i, \mathcal{D}_i^{\text{qry}}). \tag{3}$$

where α, β is the learning rate for the inner and the outer loop.

Meta-testing. After meta-training, the model is fine-tuned on the support set $\mathcal{D}_{\text{tg}}^{\text{sup}}$ of the target project P^{tg} and then tested on the query set $\mathcal{D}_{\text{tg}}^{\text{qry}}$ to evaluate the model's adaptability to the target project.

3.2 Model explainability for code summarization

Attention weight analysis. We analyze the attention weight matrices from both the code encoder and the AST encoder to understand which parts of the source code and AST sequence the model prioritizes during summary generation.

Input data mutation. Programmers often use meaningful names for methods and variables. This can cause the model to over-rely on these semantic cues,

	BLEU			METEOR			ROUGE-L		
	Baseline	Ours	↑ (%)	Baseline	Ours	↑ (%)	Baseline	Ours	↑ (%)
Target project				**Full data**					
Spring-Boot	18.82	**19.73**	4.84%	23.45	**24.08**	2.69%	48.34	**49.48**	2.36%
Spring-Framework	18.73	**19.33**	3.20%	22.39	**22.72**	1.47%	46.55	**47.05**	1.07%
Spring-Security	16.89	**17.74**	5.03%	20.70	**21.14**	2.13%	42.79	**43.61**	1.92%
Guava	31.42	**33.98**	8.15%	28.10	**29.48**	4.91%	53.96	**55.97**	3.72%
ExoPlayer	21.01	**22.30**	6.14%	23.60	**24.34**	3.14%	48.63	**49.82**	2.45%
Kafka	14.29	**15.52**	8.61%	19.10	**19.86**	3.98%	39.86	**41.27**	3.54%
Dubbo	14.91	**16.53**	10.87%	18.31	**19.46**	6.28%	38.83	**41.20**	6.10%
Flink	17.00	**17.35**	2.06%	20.92	**21.08**	0.76%	43.19	**43.55**	0.83%
Average	19.13	**20.31**	6.11%	22.07	**22.77**	3.17%	45.27	**46.49**	2.75%
# Samples				**Limited data**					
0 (zero-shot)	4.78	**5.16**	7.85%	12.71	**13.22**	3.96%	27.86	**29.77**	6.79%
50	5.83	**6.46**	10.58%	13.30	**13.84**	4.06%	29.85	**31.61**	5.94%
100	6.56	**7.27**	10.65%	13.91	**14.52**	4.35%	30.96	**32.71**	5.69%

Table 1: Comparison of performance between the baseline and our model in full data and limited data settings (↑ indicates improvement percentage).

Source Projects	BLEU	METEOR	ROUGE-L
R1 R2	8.82	17.06	38.34
R1 R2 R3	8.78	17.14	38.36
R1 R2 R3 R4	**8.96**	**17.15**	**38.80**
R2 R3 R4	8.29	16.53	37.62
R3 R4	8.56	16.81	38.09

Table 2: Results on the Spring-Boot project with different meta-datasets.

potentially limiting its generalizability to code with less informative naming. To assess this, we conduct a causal analysis by replacing function and variable names with generic terms and evaluating the effect on performance.

4 Experiment

4.1 Experimental setup

We utilize the Java dataset provided by Hu et al. [8] for pre-training and eight projects in the project-specific dataset from Xie et al. [11] for meta-learning. We perform some preprocessing steps as described in [8] with some additional steps: removing code sequences longer than 313 tokens and comments exceeding 25 tokens (based on data distribution) and filtering duplicate pairs (with an Edit Distance Ratio above 0.9). The meta-dataset includes the top three projects from the meta-dataset selection, and the fourth-ranked project used for validation. Batch sizes are set to 32 for pre-training and 16 for support and query sets in meta-learning. Inner and outer loop learning rates are set at 0.05 and 0.001, respectively. Following [11], we perform 5-fold cross-validation and simulate low-data scenarios by randomly sampling 50 or 100 training examples. Generated summaries are evaluated using BLEU, METEOR, and ROUGE-L metrics. For input data mutation experiments, we rename methods to “func” and variables sequentially to “var_i”, ensuring functional equivalence as in [13].

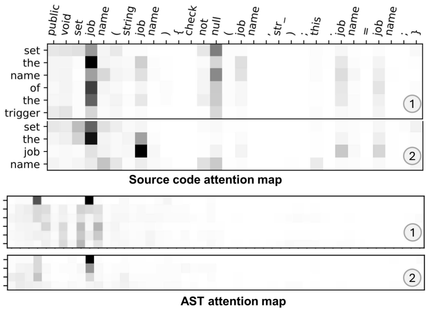

Fig. 2: Attention maps for our model (1) and the baseline (2) corresponding to code and AST sequence.

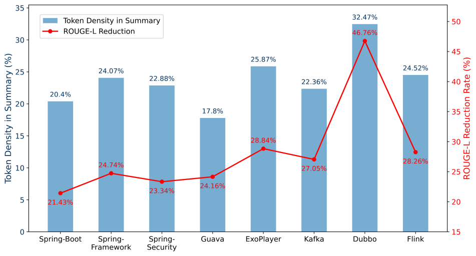

Fig. 3: Identifier density in the summaries and the drop rate in ROUGE-L scores in data mutation experiment.

4.2 Experimental results

We evaluated our MAML-based approach against conventional training in eight projects, as shown in Table 1. MAML outperforms the baseline with average improvements of 6.11% (BLEU), 3.17% (METEOR), and 2.75% (ROUGE-L). The gains are more noticeable on smaller datasets like Dubbo (10.87% BLEU improvement) but lower for larger projects like Flink. In low-data and zero-shot scenarios, our approach maintains a notable advantage, highlighting its potential to generate quality code summaries even with scarce training data.

We also assessed the impact of meta-dataset selection by training the model on Spring-Boot with four top-ranked source projects: Spring-Framework, Dubbo, Flink, and Kafka. Table 2 shows that using multiple source projects significantly boosts MAML's performance, with the best results achieved when all four are included. Importantly, high-ranked source projects (R1, R2) contribute to better outcomes, demonstrating that closely related projects enable optimal model initialization for faster and more effective adaptation. These findings highlight the importance of meta-dataset selection in enhancing model performance.

4.3 Analysis of model results

Figure 2 shows a clear difference in attention patterns between our model and the baseline. While our model focuses on key tokens like function and variable names, the baseline wrongly emphasizes trivial tokens like "null" and "job". Both models overlook structural information in SBT sequences, relying instead on identifiers like function and variable names. This observation underscores the models' dependency on semantic cues rather than the code's underlying logic. Data mutation experiment further confirms this in Figure 3, showing a marked performance decline when semantic information is obscured, with the Dubbo project suffering a 46.76% drop in ROUGE-L scores. The model performance degradation correlates with the density of identifier tokens in the summaries, emphasizing that projects rich in meaningful identifiers are more sensitive to these changes. These findings challenge researchers to develop training strategies or models enabling a deeper understanding of the code's structure and logic.

5 Conclusion

This study introduces a MAML-based training framework for code summarization in low-resource scenarios, supported by a meta-dataset selection strategy that boosts training efficiency and model adaptability. Using explainability techniques, we find that the model heavily depends on identifier semantics while underutilizing code structure. This underscores the need for future efforts to better integrate structural and semantic information of source code.

References

[1] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976, 2017.

[2] Paul W McBurney and Collin McMillan. Automatic source code summarization of context for java methods. *IEEE Transactions on Software Engineering*, 42(2):103–119, 2015.

[3] WP Li, JF Zhao, and B Xie. Summary extraction method for code topic based on lda [j]. *Computer Science*, 2017(04):42–45, 2017.

[4] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *54th Annual Meeting of the Association for Computational Linguistics 2016*, pages 2073–2083. Association for Computational Linguistics, 2016.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[6] Patrick Fernandes, Miltiadis Allamanis, and Marc Brockschmidt. Structured neural summarization. *arXiv preprint arXiv:1811.01824*, 2018.

[7] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. *arXiv preprint arXiv:2005.00653*, 2020.

[8] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation with hybrid lexical and syntactical information. *Empirical Software Engineering*, 25:2179–2217, 2020.

[9] Yi-Syuan Chen and Hong-Han Shuai. Meta-transfer learning for low-resource abstractive summarization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 12692–12700, 2021.

[10] Jiatao Gu, Yong Wang, Yun Chen, Kyunghyun Cho, and Victor O. K. Li. Meta-learning for low-resource neural machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

[11] Rui Xie, Tianxiang Hu, Wei Ye, and Shikun Zhang. Low-resources project-specific code summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, 2022.

[12] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. Retrieval-based neural source code summarization. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 1385–1397, 2020.

[13] Ankita Nandkishor Sontakke, Manasi Patwardhan, Lovekesh Vig, Raveendra Kumar Medicherla, Ravindra Naik, and Gautam Shroff. Code summarization: Do transformers really understand code? In *Deep Learning for Code Workshop*, 2022.