Coding-Fuse: Efficient Fusion of Code Pre-Trained Models for Classification Tasks

Yu Zhao[†,§], Lina Gong[†,‡,§,*], Zhiqiu Huang[†,‡,§,*], Yuchen Jin[†,¶] and Mingqiang Wei[†,¶]
[†]*School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China*
[‡]*Shenzhen Research Institute, Nanjing University of Aeronautics and Astronautics, Shenzhen, China*
[§]*Key Laboratory of Safety-Critical Software Development and Verification, Ministry of Industry and Information Technology*
[¶]*Key Laboratory of Brain-Machine Intelligence Technology, Ministry of Education*
[*]*Corresponding authors*
Email: {zhao_yu, gonglina, zqhuang, jin_yuchen, mqwei}@nuaa.edu.cn

***Abstract*—Software engineering (SE) classification tasks play a vital role in improving software quality. Nevertheless, SE researchers and practitioners tend to rely on a single code pre-trained model (PTM) for downstream classification tasks. Previous studies have found that different code PTMs yield different performance in SE classification tasks, which triggers our thinking of whether the integration of multiple code PTMs improves the performance of classification tasks. Therefore, we first conduct preliminary exploratory research to analyze the impact of fusing multiple PTMs on code classification tasks. The result shows that compared to the single code PTM, the fusion of multiple code PTMs can improve the performance of SE classification tasks. However, the performance improvement also brings about the problem of increased finetuning resources and reduced application efficiency, which does not meet the greenness requirements. In order to address these issues, we propose Coding-Fuse, a framework of efficient fusion of code PTMs for SE classification tasks. Coding-Fuse first introduces evidence theory to evaluate the adaptability of the output features of each layer of code PTMs and data labels, and locates the potential best performance layer of different code PTMs. Then, Coding-Fuse uses a soft voting strategy to fuse the outputs of these layers to obtain a new model. We conduct experiments for effectiveness by comparing Coding-Fuse with the full PTM fusion method and the original single PTM using five different code PTMs on three different SE classification tasks and two task scenarios. The results show that Coding-Fuse can achieve better performance than the full PTM fusion method with higher efficiency and fewer hardware resources, and can achieve better performance than the original single PTM at the same efficiency and hardware resource level. We encourage SE practitioners to use our Coding-Fuse method in practice to fully utilize the advantages of each code PTM in the PTM repository according to task requirements to easily create new SE intelligent PTMs to achieve performance and greenness improvements.**

***Index Terms*—Coding-Fuse, Code Pre-trained Model Fusion, Maximum Evidence, Layer Fusion, Soft Vote, Performance and Greenness.**

I. Introduction

Research related to SE classification tasks has always been an important topic in the community. Improving the performance of these tasks can not only help improve software development efficiency, but also enhance code quality control and automation support [1], [2]. For example, code clone detection tasks to refactor and reduce redundancy, just-in-time software defect prediction tasks to prevent code commits from introducing bugs, etc. With the significant advantages of code PTMs in program semantic understanding and structural modeling [3], [4], SE researchers and practitioners generally use code PTMs to improve the performance of code-related classification tasks [5], [6].

However, in actual research and application, SE researchers and practitioners tend to rely on a single code PTM to generate code embeddings or finetune it to adapt to specific downstream classification tasks. Is this the most appropriate way to use the code PTMs?

Previous research [7], [8] found that due to differences in pre-training corpora, different code PTMs perform differently in SE classification tasks and show different advantages on the same task. This practice of using only one single code PTM limits the in-depth understanding and full use of different model capabilities. Inspired by the idea of ensemble learning, **is it possible that using multiple code PTMs simultaneously will further improve the performance of code classification tasks?** At present, no one has verified this issue. Different from prior studies exploring ensemble learning on different ML algorithms (e.g., Decision Tree and Random Forest) for code tasks [9], we first systematically conduct an empirical study to analyze that compared to the original single PTM, whether the fusion model of different code PTMs with architectures through the ensemble idea will improve the performance of the code-related classification task.

Specifically, we use five different PTMs (including BERT, CodeBERT, StarEncoder, PLBART, and CodeT5) to conduct large-scale experiments on the code clone detection task in two task scenarios: code embedding scenario and finetune model scenario. We observe that compared with the original single model, the fusion of multiple full code PTMs can improve the performance of code clone detection task in both the code embedding scenario and the finetune model scenario. However, another problem appears, that is, directly fusing models leads to a surge in model parameters, which will bring additional costs in the process of finetuning and applying. This inspired us to propose a solution for the first time that whether we can reduce the parameters of the code PTMs without affecting the performance.

Recently, many studies [10]–[12] have explored the repre-

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00157

sentation of code PTMs for classification tasks. They found that compared with the use of the last hidden layer, i.e., the full code PTM, the effect of using only the early layers of the code PTM for code-related classification tasks is equally good. For example, Grishina et al. [12] found that only the early layers of CodeBERT can achieve the effect of using the full model on vulnerability detection tasks.

Inspired by these works, we realize that for SE classification tasks, the full code PTM can only retain a few early layers without affecting the task performance. Based on this, this paper proposes Coding-Fuse, a framework of efficient fusion of code pre-trained models for classification tasks, which is very simple and consists of two steps. The first step is to locate the early layers mentioned above that are effective for classification tasks, which are called the potential best layers. The second step is to fuse these early potential best layers of different code PTMs to obtain a new code PTM. Specifically, we first apply evidence theory to estimate the compatibility between features and labels, that is, to calculate the maximum evidence value between the output features of each layer of the code PTM and the data label. The layer with the largest maximum evidence value is the layer with the best potential performance. Then, we use the soft voting strategy to fuse the outputs of these layers to obtain a new code PTM that combines the strengths of different existing code PTMs.

To verify the effectiveness of the proposed method, we use the same as mentioned above five different code PTMs to conduct experiments on three different code-related classification tasks in the two scenarios. The results show that the proposed Coding-Fuse method can achieve better F1-Score and AUC-PR than the full model fusion method with higher efficiency and fewer hardware resources and also can achieve better F1-Score and AUC-PR than the original single model at the same efficiency and hardware resource level on the code-related classification tasks in both the code embedding scenario and the finetune model scenario.

Furthermore, we discuss the reasons for the effectiveness of Coding-Fuse. We find that the average entropy of the feature vectors output by Coding-Fuse is larger than that of the full model fusion method, which means that the feature vectors output by Coding-Fuse contain more information. This result shows that the feature vectors output by Coding-Fuse provide richer information, which in turn leads to the performance advantage of the Coding-Fuse method over the full model fusion method in classification tasks.

This paper provides guidance for future SE researchers and practitioners, that is, they can focus only on the early to middle layers when using code PTMs for downstream classification tasks, and they can apply the new paradigm of multiple code PTMs fusion to code classification tasks, and try to use the Coding-Fuse framework to generate a new fused code PTM, so as to fully combine and utilize the advantages of various code PTMs in the open source repository to improve performance, efficiency and resource utilization.

In summary, the main contributions of this paper are as follows:

- We systematically investigate the impact of fusing multiple code PTMs compared with the original single code PTM on the SE classification tasks.
- We propose Coding-Fuse, a framework of efficient fusion of code PTMs for SE classification tasks that can locate the potential best layers of different code PTMs and fusion them into a new model to combine the strengths of individuals.
- We demonstrate the effectiveness of the proposed method Coding-Fuse in terms of performance and greenness through large-scale experiments.
- We make our code and data publicly available in the open-source repository to promote community research and practice at https://github.com/ASE-2025.

Paper Organization. For the remainder of this paper, Section II discusses the related works. Section III presents candidate code PTMs and the experimental tasks and datasets used in our study. Section IV provides the motivation, approach, and results in the preliminary exploratory study. Section V provides the details of the proposed method Coding-Fuse. Section VI evaluates the effectiveness of the proposed method Coding-Fuse in performance and greenness evaluation. Section VII discusses the reasons for the effectiveness of the Coding-Fuse and additional implications for SE practitioners, as well as the validity threats to our experiment. Finally, Section VIII concludes the paper.

II. Related Work

This section introduces related work from two aspects. The first is the application of code PTMs to code-related classification tasks, and the second is the exploration and analysis of code pre-trained models.

A. Code PTMs applied to SE classification tasks

With the popularity of PTM and its remarkable achievements in natural language (NL) understanding and generation tasks [3], [4], many SE researchers and practitioners have also considered using code PTMs when solving code-related classification tasks [5], [6]. For example, in view of the low efficiency of pairwise comparison of code pairs in existing clone detection methods and poor scalability on industrial systems, Ahmed et al. [13] proposed a BERT-based clone detection method, which aims to achieve high Recall for Type-3 and Type-4 clones and can be applied to large software systems. To further improve the accuracy of just-in-time software defect prediction, Ni et al. [14] proposed a CodeBERT-based method by integrating semantic features and expert features, achieving improvements in multiple metrics of file-level and line-level predictions. Considering that all large manually labeled datasets in the field of code smell detection are developed for Java, Aleksandar et al. [15], [16] created a new C# dataset to alleviate this problem and used CodeT5 to detect code smells in C#. In addition, the same is true for areas that are closely related to practice. Zeng et al. [17] manually labeled issue reports from 11 deep learning projects to construct a new bug issue type dataset

TABLE I
FIVE CANDIDATE CODE PRE-TRAINED MODELS.

PTM Name	Architecture	Size	Dimension	Pre-training Corpus
BERT	Encoder	110M	768	3300M pure English text words from BooksCorpus and Wikipedia
CodeBERT	Encoder	125M	768	8.5M datapoints across 6 programming languages from Github repositories
StarEncoder	Encoder	125M	768	86 programming languages from GitHub issues and Git Commits
PLBART	Encoder-Decoder	140M	768	650GB of Java and Python functions and text descriptions from GitHub and StackOverflow
CodeT5	Encoder-Decoder	220M	768	8.35M instances across 8 programming languages from CodeSearchNet and BigQuery

and developed an automated classification method based on BERT that can accurately identify the bug issue types of deep learning projects. Le et al. [18] proposed a novel semantic-based log parsing method and found that a semantic-based log parser using a small PTM can actually achieve better or comparable performance than the state-of-the-art LLM-based log parsing model, while being more efficient and cost-effective.

By investigating past studies, we found that SE researchers and practitioners tend to use only one single code PTM to generate code embeddings or finetune when solving code-related classification tasks. This practice would limit the in-depth understanding and full utilization of diverse code PTMs capabilities, which motivates us in this work to investigate the possibility of using multiple models simultaneously in the code classification task.

B. Code PTMs exploration and analysis

Many researchers have explored and analyzed the code PTMs to decipher the internal mechanism of code PTMs that understand the code. Karmakar et al. [19], [20] used probe tasks to explore how well code PTMs learn from specific aspects of source code. They used an extensible framework to define 15 probe tasks to analyze 8 code PTMs about the surface, syntax, structure and semantic features of the source code. Ma et al. [21] conducted a similar study that analyzed the ability of code PTMs to learn code grammar and semantics, i.e., understand the information of code grammar and semantic structures such as AST, CDG, DDG, and CFG in the representation space. Wan et al. [22] further analyzed the code PTMs from three unique perspectives: attention mechanism analysis, word vector exploration, and syntax tree induction. Grishina et al. [12] conducted an empirical study of code PTMs from the layer perspective. They proposed a general method EarlyBIRD, which is used to build a composite representation of code from the early layers of the code PTMs. The feasibility of this approach is empirically studied on CodeBERT by comparing the performance of 12 strategies for constructing composite representations with the standard practice of using only the last layer. Shi et al. [11] also focused on the probe of each layer of the code PTMs, mainly exploring what changes will occur in the pre-trained representation and its code knowledge during the finetuning process. They found that the vocabulary, syntactic, and structural properties of the source code were encoded in the lower, middle and higher layers, respectively, while the semantic properties ran through the entire model, and the basic code properties captured by the lower and middle layers during the finetuning process were still retained during the fine-tuning process.

These works on the exploration of code PTMs, especially the exploration of layers, inspired us to consider finding the most relevant layers of different code PTMs to fuse and apply to downstream classification tasks and implement new paradigms for constructing code PTMs.

III. EXPERIMENTAL DATA

In this section, we provide a detailed description of the key components involved in the experiment of our study, including the candidate code PTMs utilized for model fusion and the three code-related classification task and datasets employed for experimental research.

A. Candidate code pre-trained models

To ensure the comprehensiveness of the experiments in our study and the applicability of the conclusions, we examine five different code PTMs (more than two PTMs for each architecture) with different backgrounds, that is, the pre-training corpus corresponding to these models should be diversified, so as to integrate different advantages. Details can be seen in Table I, we select BERT [23], CodeBERT [24], StarEncoder [25], PLBART [26] and CodeT5 [27]. Note that 1) Although BERT is not a code PTM, we would choose it because many people have used BERT for SE classification tasks in the past and the results are very good [13], [17] and recent exploratory studies [19], [20] have also shown that BERT has a good ability to understand code and should become a benchmark in the SE field. 2) We choose popular encoder-only, encoder-decoder and exclude decoder-only architectures code PTMs since code PTMs with encoder-only and encoder-decoder architectures perform well in SE classification tasks, decoder-only architectures are designed for generation tasks and may difficult to show advantages in classification tasks [3], [28].

B. Code-related classification tasks and datasets

In order to ensure the generalizability of the conclusions drawn in the experiment, we choose three classification tasks in the SE field: code clone detection (CCD), code smell detection (CSD), and technical debt detection (TDD).

These classification tasks are important and popular tasks that rank high in the SE field, and have always dominated the mainstream of research in the SE field [29]–[31]. As can be seen, Table II presents brief information on the studied datasets. For the CCD task, we conduct experiments on the widely recognized BigCloneBench dataset [32], this dataset

TABLE II
STATISTICS OF THE THREE CLASSIFICATION TASKS DATASETS.

Task Name	Examples Num	Train/Valid/Test	Main Content Type	Token Length Distribution
Code Clone Detection	1731860	901028/415416/415416	Clone Type-1, Type-2, Type-3, Type-4	78%≤400, 22%>400
Code Smell Detection	1789	1089/350/350	Misleading Identifier Names/Naming Convention Violations	96%≤50, 4%>50
Technical Debt Detection	38360	24034/7674/6652	Design Debt/Requirement Debt	94%≤200, 6%>200

contains four different types of Java code clones [33] from Type-1 to Type-4. In the case of the CSD task, we utilize the corpus shared by Fakhoury et al. [34], where the code smells come from the use of misleading identifier names or violations of common naming conventions [35]. For the last TDD task, we use the dataset shared by Maldonado et al. [36], whose source code comments were extracted from Java projects GitHub to detect self-admitted technical debt [37]. These chosen tasks and datasets display diversity to get stable conclusions and ensure that can be generalized across a broader spectrum.

IV. PRELIMINARY STUDY

The current mainstream for SE researchers and practitioners is to choose appropriate code PTMs across multiple programming languages, architectures, and application scenarios and integrate them into the pipeline of constructing deep learning-based classification task models to save training costs and improve classification performance. However, previous studies [7] have found that different code PTMs have differentiated performance in SE downstream classification tasks, which means that different models have different strengths for the same task. In this case, is it still a good choice to use only one specific code pre-trained model? More generally, when new classification task scenarios emerge, is it possible to improve the classification performance by integrating multiple existing PTMs so as to combine the advantages of different PTMs for the same classification task? To address this issue, in this section, we will construct a preliminary study to understand the impact of fusing multiple PTMs on SE downstream classification tasks compared to the original single PTM. More specifically, we will proceed by answering the following question:

RQ1: What is the impact of fusing multiple PTMs on SE downstream classification tasks compared to the original single PTM?

Approach: In order to tackle this issue, we conduct simple pairwise fusion experiments on the code clone detection task using five different code PTMs shown in Table I and systematically compare the performance and greenness of full model fusion and the original single model in the code embedding scenario and the finetune model scenario. The reason for choosing the code clone detection task for exploratory experiments is that the total sample size of the dataset corresponding to this task exceeds 1.7 million, and it contains high-quality Java function codes, which generally contain rich information and are sufficient to represent the results of the SE classification tasks.

Specifically, for the original single model applied to the CCD task, we follow the design pipeline [38] proposed in previous studies to connect the fully connected layer classifier after the code PTMs. For the code embedding scenario, i.e., code PTMs are used as fixed feature extractors for code snippets or structures, with all their parameters frozen and a separate task model trained afterward, as in [39], we also freeze the parameters of the code PTMs and only finetune the classifier layer. For the finetune model scenario, i.e., code PTMs are integrated into the end-to-end task model pipeline, with all parameters to be updated during training, as in [40], we also finetune all parameters of the code PTMs and the classifier layer. For the full model fusion method applied to the CCD task, the experiment was also divided into two scenarios according to the above design pipeline. The difference is that we use two different entire PTMs for encoding and finetuning at the same time, and finally fuse the prediction results at the classifier layer. It is necessary to add that to ensure adequate training of the model, we set a maximum of 100 iterations for all experiments and when the F1-Score value on the validation set did not increase for 10 consecutive rounds, we stopped training and saved the model weight with the best F1-Score value on the validation set.

In order to conduct a thorough evaluation, for the performance evaluation, we evaluate the F1-Score [41], [42] and AUC-PR [43]–[45] values of the models on the test dataset in the two scenarios. The previous is the preferred threshold performance metric, while the latter is a threshold-independent performance metric that takes into account the dataset imbalance. Both of which the larger the value, the better the performance. Combining the two can make a more comprehensive evaluation of the performance. For the greenness evaluation [46], we evaluate the throughput and GPU memory occupied during training in the finetune model scenario, while considering the latency time during model prediction and the final storage consumption of the model. Training throughput (example/second) measures how quickly data can be processed during training. Higher throughput reduces training time, leading to lower energy consumption and carbon emissions. GPU memory usage (MB) indicates the hardware footprint during training. Models requiring less memory can be trained on smaller, more energy-efficient GPUs, thereby reducing both electricity and cooling costs. Test latency (second) is critical in deployment, especially on edge devices or in real-time systems. Lower latency implies faster response and reduced power usage. Model storage size (MB) determines the resources needed for storing and transferring the model. Smaller models are easier to deploy

TABLE III
COMPARISON RESULTS OF MULTIPLE PERFORMANCE AND GREENNESS EVALUATION METRICS OF THE OSM AND THE FMF ON THE CCD TASK. ↑ INDICATES THE LARGER THE BETTER.

Settings	Scenarios	Code embedding scenario		Finetune model scenario					
	Metrics	F1-Score↑	AUC-PR↑	F1-Score↑	AUC-PR↑	GPU (MB)↓	Throughput (example/s)↑	Latency (s)↓	Storage (MB)↓
Original Single PTM (OSM)	BERT	0.602	0.638	0.901	0.907	16599	56	6.83	418
	CodeBERT	0.716	0.735	0.931	0.935	17015	54	8.86	476
	StarEncoder	0.723	0.747	0.915	0.920	17013	55	8.61	474
	PLBART	0.446	0.613	0.919	0.923	28676	42	8.76	531
	CodeT5	0.511	0.570	0.949	0.952	54499	27	10.27	850
Full PTM Fusion (FMF)	CodeBERT+BERT	**0.729**	**0.753**	*0.911*	*0.916*	*34220*	*28*	*9.02*	*893*
	CodeBERT+StarEncoder	**0.776**	**0.792**	**0.937**	**0.941**	*34580*	*28*	*9.51*	*949*
	CodeBERT+PLBART	**0.717**	**0.742**	**0.940**	**0.943**	*43774*	*25*	*9.51*	*1007*
	CodeBERT+CodeT5	**0.727**	**0.745**	*0.947*	*0.950*	*75534*	*17*	*10.99*	*1321*
	BERT+StarEncoder	**0.739**	**0.756**	**0.939**	**0.942**	*34340*	*29*	*9.19*	*892*
	BERT+PLBART	*0.544*	*0.626*	**0.943**	**0.947**	*43448*	*25*	*8.26*	*949*
	BERT+CodeT5	*0.528*	*0.590*	**0.952**	**0.954**	*75168*	*17*	*12.76*	*1268*
	StarEncoder+PLBART	**0.734**	**0.761**	**0.931**	**0.936**	*43744*	*25*	*9.70*	*1005*
	StarEncoder+CodeT5	*0.719*	*0.743*	**0.953**	**0.956**	*75560*	*17*	*11.30*	*1324*
	PLBART+CodeT5	**0.546**	**0.651**	**0.952**	**0.956**	*86308*	*16*	*12.28*	*1382*

across platforms and consume less disk space and bandwidth. These metrics reflect both the efficiency of the training process and the environmental cost of real-world deployment. These four are also commonly used metrics in related fields, which can achieve exhaustive evaluations of greenness in actual deployment [47].

Furthermore, in order to ensure the fairness of the experiment itself, we divide the dataset into training set, validation set and test set in the same proportion as previous studies [38]. In addition, we also control the input information of each model to be the same, that is, the length of the input code of the CCD task is controlled at 400 tokens. As shown in Table II, this is enough to cover most of the samples in the dataset, ensuring the amount of data information while avoiding the influence of too many padding tokens on the experimental results [48]–[50], ensuring the adequacy of the experimental results.

Results: Table III shows the various evaluation metrics of the performance evaluation obtained on the test set of the code clone detection task based on the original single five different code PTMs (referred as OSM) and the full PTM fusion (referred as FMF) as well as the greenness evaluation during the model training process. Among them, the bold values indicate that FMF is better than all the corresponding two OSMs, and the non-bold italic values indicate that FMF is worse than one or all of the corresponding two OSMs.

Observation 1) In both code embedding scenario and finetune model scenario, the FMF method has performance advantages over the corresponding original single models and can improve the F1-Score and AUC-PR of the code clone detection task. As shown in the CCD task in Table III, for the code embedding scenario, compared with the original single PTMs method, the code embedding generated by fusing two full PTMs and then finetuning the classifier achieves better F1-Score and AUC-PR values for seven times. Among them, the highest performance improvement is the fusion of StarEncoder and PLBART, which improves F1-Score and AUC-PR by most 65% and 24% respectively compared to the two single PTMs. In only three cases, the FMF method does not completely outperform the two corresponding single PTMs before fusion. While the FMF method in these three cases does not lead to worse performance, and the performance value after fusion was between the performance values of the corresponding two original single PTMs before fusion, which ensures the lowest performance limit. For the finetune model scenario, the results are also the same. The difference is that fusing two full PTMs achieves better F1-Score and AUC-PR values for eight times and the remaining two cases of performance between the corresponding two original single PTMs. In this scenario, the biggest performance improvement is the fusion of BERT and CodeT5, which increases F1-Score and AUC-PR by 6% and 5% respectively. The superior performance of the FMF method over OSM in both scenarios shows that fusing multiple full PTMs is a promising approach for code-related downstream classification tasks.

Observation 2) The hardware resources required for training and testing the FMF method far exceed those of the corresponding original single models in the finetune model scenario, and its efficiency is also far worse than them. As shown in Table III, it can be seen that compared with the original single PTMs, the GPU resources required for finetuning all parameters and the final model storage space occupied by FMF mothod are almost equal to the sum of the GPU memory required for finetuning and storage space for storing each of the corresponding original single PTMs. This situation holds true for all the experiments in the ten cases. In addition, it can also be seen that the throughput of the FMF method during training is lower than that of the corresponding original single model, and the latency during testing is also higher than that of the corresponding original single model, which is also a common phenomenon in the ten cases. The higher hardware resources and lower operating efficiency of the FMF method compared to OSM indicate that fusing multiple full PTMs may not meet practical needs in

actual deployment.

In terms of comprehensive performance and greenness aspects, for downstream classification tasks related to code such as code clone detection, although the simple fusion of multiple full PTMs achieves superior performance compared to the original single PTM, the cost of this performance improvement is the doubling of hardware training resources and storage resources as well as lower training and testing efficiency compared to the original single PTM. Considering the performance and greenness trade-offs in actual deployment, the current simple full PTM fusion method may not meet practical needs, and there is still huge room for improvement.

Summary. Compared with the OSM, the FMF method can improve the performance of SE classification tasks in both the code embedding scenario and the finetune model scenario, which seems promising. However, simple FMF method is difficult to bring out the advantages of each individual sufficiently on the one hand, i.e., limited performance gain, and the far more hardware resources and much lower efficiency than the OSM in the finetune model scenario, which makes it difficult to deploy in practice, i.e., unmet greenness need.

V. Coding-Fuse: Efficient Fusion of Code Pre-Trained Models for Classification Tasks

Previous studies [11], [12] have shown that the effect of finetuning the early layers of code PTMs for SE classification tasks exceeds that of using the full code PTMs. Inspired by these works, we realize that by fusing the early layers of these code PTMs, we can combine the advantages of different models and achieve performance improvement while solving the difficulties in actual deployment. However, one of the difficulties in achieving this goal is how to determine the early layers with the best potential performance. To this end, this paper proposes Coding-Fuse, a framework of efficient fusion of code pre-trained models for classification tasks. As shown in Figure 1, the core part of the framework is very simple, divided into identifying potential best layer module and fusing potential best layer module. The identifying potential best layer module is to estimate the performance of the output features of all layers of each code PTM on the SE classification task by introducing evidence theory and locate the layer with the potential best performance. The fusing potential best layer module is to fuse the previously located layers together according to actual needs to form a new model and apply it to the downstream classification task. And finally achieve the goal of improving performance while keeping greenness for actual deployment. Below we will explain the framework in detail.

A. Identify potential best layer

First, define the problem scenario. For a code PTM $\mathcal{M}$ with L layers, i.e. $\mathcal{M} = \{\phi_l\}_{l=1}^{L}$, given a target classification task and a corresponding dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where a portion of n instances are labeled, the goal is to find a specific layer l^* of the PTM $\mathcal{M}$ such that the sub-part of the code PTM up to l^* has the best classification performance on the dataset $\mathcal{D}$ after appropriate hyperparameter tuning, as measured by some performance evaluation metrics, such as Accuracy or F1-Score. For each layer ϕ_l of the code PTM $\mathcal{M}$, a practical evaluation method should produce a score S_l for ϕ_l and ideally does not require additional effort for finetuning on $\mathcal{D}$. In this way, the corresponding layer l^* of the code PTM with the best potential performance can be selected by simply comparing the scores. To achieve this, we avoid finetuning all the parameters of the model. That is, after inputting data into the code PTM, the features F extracted by each layer of the frozen model and the corresponding data labels y are used for evaluation. In this case, the problem becomes estimating the compatibility between features and labels.

Second, calculate and maximize the evidence of each layer's output feature with the label. The most directly related compatibility measure between the features F extracted at each layer L and the corresponding data labels y is the probability density $p(y|F)$. For further processing, according to the empirical practice of applying code PTMs to code classification tasks, a fully connected layer is added to associate features and labels, which can be parameterized as w by a linear model, as shown in Figure 2. One solution to deal with linear models is to find

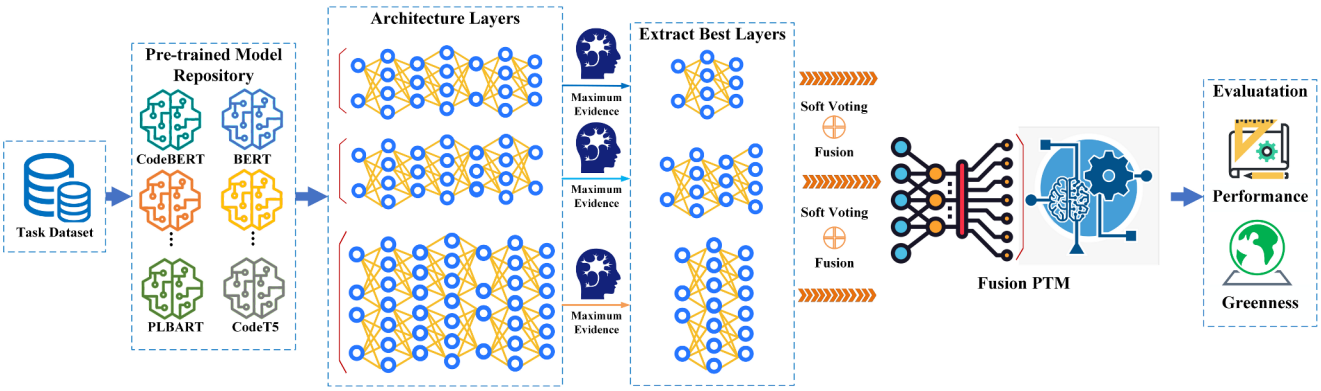

Fig. 1. The framework of the proposed Coding-Fuse.

an optimal w^* matrix through the maximum likelihood function $p(y|F,w)$. However, the maximum likelihood is prone to overfitting, that is, the w^* found is inaccurate. Therefore, we use the evidence theory, i.e., marginalized likelihood, $p(y|F) = \int p(w)p(y|F,w)\mathrm{d}w$, which was proposed by Knuth et al. [51], optimized by You et al. [52], and successfully applied to the scenario of model transfer learning, which has a rigorous theoretical basis. Specifically, the evidence $p(y|F) = \int p(w)p(y|F,w)\mathrm{d}w$ is the integral with respect to w, which considers all values w and is better than simply using the optimal value w^*.

For the analytical solution of evidence, we also adopt the scheme of You et al [52]. Specifically, assume that w and y_i satisfy the Gaussian distribution $w \sim \mathcal{N}(0, \alpha^{-1}I)$ and $y_i \sim \mathcal{N}(w^T f_i, \beta^{-1})$ respectively, for $p(y|F,w)$, the distribution of each observation is a one-dimensional normal distribution $p(y_i|f_i,w,\beta) = \mathcal{N}(y_i|w^T f_i, \beta^{-1})$. Then the evidence can be calculated as the next equation.

$$
\begin{aligned}
p(y|F,\alpha,\beta) &= \int p(w|\alpha)\, p(y|F,w,\beta)\, \mathrm{d}w \\
&= \int p(w|\alpha) \prod_{i=1}^{n} p(y_i|f_i,w,\beta)\, \mathrm{d}w \\
&= \left(\frac{\beta}{2\pi}\right)^{\frac{n}{2}} \left(\frac{\alpha}{2\pi}\right)^{\frac{D}{2}} \int e^{-\frac{\alpha}{2}w^T w - \frac{\beta}{2}\|Fw-y\|^2}\, \mathrm{d}w
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \log p(y|F,\alpha,\beta) = {} & \frac{n}{2}\log\beta + \frac{D}{2}\log\alpha - \frac{n}{2}\log 2\pi \\
& - \frac{\beta}{2}\|Fm-y\|_2^2 - \frac{\alpha}{2}m^T m - \frac{1}{2}\log|A|
\end{aligned}
$$

where

$$A = \alpha I + \beta F^T F, m = \beta A^{-1} F^T y$$

To determine the values of α, β when the evidence is maximized, according to the derivation of Gull et al. [53] and You et al. [52], maximizing the evidence $\log p(y|F,\alpha,\beta)$ can be done by alternating between evaluating $m = \beta A^{-1} F^T y, \gamma = \sum_{i=1}^{D} \frac{\beta\sigma_i}{\alpha+\beta\sigma_i}$ and maximizing $\alpha \leftarrow \frac{\gamma}{m^T m}, \beta \leftarrow \frac{n-\gamma}{||Fm-y||_2^2}$ with m, γ fixed, where $\{\sigma_i\}_{i=1}^{D} = \mathrm{SVs}(F^\top F)$. The final achieved $S_l = \log p(y|F,\alpha^*,\beta^*)$ can evaluate the compatibility between the layer's output features and labels.

Finally, locate the layer with the largest maximum-evidence value. We input data into the code PTM $\mathcal{M}$, calculate the maximum evidence $\{S_l\}_{l=1}^{L}$ for the output features and corresponding data labels of each layer $\{\phi_l\}_{l=1}^{L}$, compare and select the corresponding l layer with the largest maximum evidence value $Max\{S_l\}_{l=1}^{L}$, and consider it as the best potential performance of the entire model. This means that when loading the model, all weights from this layer to the last layer are discarded, thus achieving a balance between performance and greenness.

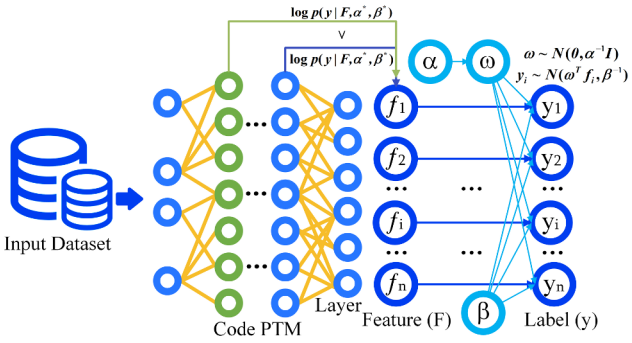

Fig. 2. The illustration for calculating evidence.

B. Fuse potential best layer

After identifying the optimal representation layer for each code PTM, we extract the hidden states from that specific layer in each code PTM and subsequently fuse them from both code PTMs. We employ a relatively simple yet effective output-level soft voting strategy, where the classification probabilities are averaged to get a fusion. We do this instead of concatenating or averaging the feature representations since the vocabulary differences between code PTMs hinder direct feature fusion. Specifically, two distinct code PTMs are utilized to encode the input sequence, from which we extract the hidden states at their respective optimal layers. Following a commonly adopted practice in extracting representation [38], [54], we take the hidden vector corresponding to special tokens (e.g., [CLS]) as a global semantic representation of the entire input. The above soft voting strategy is then applied to obtain the fusion result.

VI. Experimental Validation

In this section, we design the next two research questions (RQs) to verify the effectiveness of the proposed method Coding-Fuse. Below we will elaborate on the approach and result of these two RQs to facilitate experimental verification and result analysis.

RQ2: How does Coding-Fuse perform in performance?

RQ3: How does Coding-Fuse perform in greenness?

Approach: To address these two issues and for a more comprehensive assessment, we use the five different code PTMs shown in Table I to conduct experimental comparisons on the code-related classification tasks, which are expanded to the three tasks shown in Table II. And similar to the previous exploratory study in Section IV, we also use the same design pipeline proposed by previous researchers. In addition, we also conduct experiments on two application scenarios, the finetune settings, training strategy and performance and greenness evaluation metrics are the same as Section IV. For the additional two classification tasks shown in Table II, we also control the input information of each model to be the same, that is, the input length of the CSD task is controlled at 50 tokens, and the TDD task is controlled at 200 tokens, to ensure the fairness of the experiment itself. It is worth adding that when using the maximum evidence to find the best

TABLE IV
COMPARISON RESULTS OF MULTIPLE PERFORMANCE AND GREENNESS EVALUATION METRICS OF CODING-FUSE AND FMF ON THE CCD TASK IN CODE EMBEDDING SCENARIO AND FINETUNE MODEL SCENARIO. ↑ INDICATES THE LARGER THE BETTER.

Settings	Scenarios	Code embedding scenario		Finetune model scenario					
	Metrics	F1-Score↑	AUC-PR↑	F1-Score↑	AUC-PR↑	GPU (MB)↓	Throughput (example/s)↑	Latency (s)↓	Storage (MB)↓
Full PTM Fusion (FMF)	CodeBERT+BERT	0.729	0.753	0.911	0.916	34220	28	9.02	893
	CodeBERT+StarEncoder	0.776	0.792	0.937	0.941	34580	28	9.51	949
	CodeBERT+PLBART	0.717	0.742	0.940	0.943	43774	25	9.51	1007
	CodeBERT+CodeT5	0.727	0.745	0.947	0.950	75534	17	10.99	1321
	BERT+StarEncoder	0.739	0.756	0.939	0.942	34340	29	9.19	892
	BERT+PLBART	0.544	0.626	0.943	0.947	43448	25	8.26	949
	BERT+CodeT5	0.528	0.590	0.952	0.954	75168	17	12.76	1268
	StarEncoder+PLBART	0.734	0.761	0.931	0.936	43744	25	9.70	1005
	StarEncoder+CodeT5	0.719	0.743	0.953	0.956	75560	17	11.30	1324
	PLBART+CodeT5	0.546	0.651	0.952	0.955	86308	16	12.28	1382
Coding-Fuse	CodeBERT-3+BERT-4	**0.734**	**0.767**	**0.949**	**0.953**	**10711**	**91**	**6.60**	**438**
	CodeBERT-3+StarEncoder-2	**0.855**	**0.864**	**0.946**	**0.949**	**8777**	**118**	**8.31**	**445**
	CodeBERT-3+PLBART-4	**0.821**	**0.836**	**0.944**	**0.948**	**15629**	**78**	**7.76**	**502**
	CodeBERT-4+CodeT5-8	**0.750**	**0.778**	**0.954**	**0.957**	**21757**	**51**	**7.85**	**578**
	BERT-4+StarEncoder-2	**0.835**	**0.845**	**0.942**	**0.947**	**9589**	**103**	**6.65**	**409**
	BERT-3+PLBART-4	**0.792**	**0.807**	*0.934*	*0.938*	**15491**	**78**	**6.22**	**440**
	BERT-3+CodeT5-6	**0.774**	**0.788**	*0.950*	**0.955**	**17493**	**64**	**6.39**	**439**
	StarEncoder-2+PLBART-4	**0.842**	**0.852**	**0.950**	**0.954**	**14365**	**87**	**7.50**	**473**
	StarEncoder-3+CodeT5-6	**0.817**	**0.833**	*0.950*	*0.954*	**17589**	**65**	**7.65**	**496**
	PLBART-4+CodeT5-6	**0.802**	**0.816**	**0.953**	**0.956**	**26376**	**54**	**7.53**	**526**

layer for each model, we select the layer whose maximum evidence value is greater than the last layer of the model and corresponds to the largest first two values, and then fuse the models pairwise. In this way, there are four combinations for the fusion of any two pre-trained models. We select the combination with the best performance for comparison.

Results: Tables IV to VI show the various evaluation metrics of the proposed efficient fusion PTM method Coding-Fuse and the full PTM fusion method FMF on the three classification tasks of code clone detection (CCD), technical debt detection (TDD) and code smell detection (CSD) in the code embedding scenario and the finetune model scenario. Among them in the table, the bold values indicate that Coding-Fuse is better than FMF, and the non-bold italic values indicate that Coding-Fuse is worse than FMF. The number following the code PTM in the Coding-Fuse cell represents the extracted potential best layer of the model. For example, CodeBERT-3+BERT-4 in Table IV refers to the selected best layer number of the CodeBERT and BERT on the CCD task. Note in Table VI, since the maximum evidence values of each layer in the StarEncoder on the CSD task are smaller than the value of the last layer, this means that the potential best layer is the full model, so we excluded this model from this task due to the greenness requirements. Finally, observation 3 corresponds to the answer to RQ2, and observation 4 corresponds to the answer to RQ3.

Observation 3) The proposed efficient fusion code pre-trained model method Coding-Fuse has advantages over the full code pre-trained model fusion method FMF in terms of performance and can achieve better F1-Score and AUC-PR on the code-related classification tasks in both the code embedding scenario and the finetune model scenario. In more detail, as shown in Table IV, in the code embedding scenario of the CCD task, compared with FMF method, the Coding-Fuse method has an overwhelming advantage in F1-Score and AUC-PR values, and is better in all ten cases, with the F1-Score and AUC-PR maximum improvements of up to 46.9% and 33.6% respectively. In the finetune model scenario of the CCD task, compared to FMF method, the Coding-Fuse method also achieves great advantages in F1-Score and AUC-PR values. Seven out of ten cases achieve better values, only two cases are worse, and one case can be considered evenly matched, with the F1-Score and AUC-PR increased by up to 4.2% and 4.1% respectively. Similar results can also be found in Table V. As shown in Table V, in the code embedding scenario of the TDD task, compared with FMF method, the Coding-Fuse method achieves advantages in eight cases, and of the remaining two cases, only one is worse, and the other case can be considered as the same level, with the F1-Score and AUC-PR maximum improvements of up to 35.4% and 23% respectively. And in the finetune model scenario of the TDD task, the Coding-Fuse method achieves better performance in seven of the ten cases than FMF method, which is the same as the CCD task, that the other two cases are worse, and one case can be considered evenly matched, with the F1-Score and AUC-PR increased by up to 1.1% and 1.3% respectively. Finally, the Coding-Fuse method achieves the most excellent results in the CSD task in Table VI, which has an overwhelming advantage over the FMF method in all cases, with the F1-Score and AUC-PR maximum improvements of up to 15%, 11.6%, 24.8%, and 16.7% respectively. The outstanding results of RQ2 show that if the advantages of fusing different code PTMs are considered, the Coding-Fuse method is more effective than the FMF method. Considering

TABLE V

COMPARISON RESULTS OF MULTIPLE PERFORMANCE AND GREENNESS EVALUATION METRICS OF CODING-FUSE AND FMF ON THE TDD TASK IN CODE EMBEDDING SCENARIO AND FINETUNE MODEL SCENARIO. ↑ INDICATES THE LARGER THE BETTER.

Settings	Scenarios	Code embedding scenario		Finetune model scenario					
	Metrics	F1-Score↑	AUC-PR↑	F1-Score↑	AUC-PR↑	GPU (MB)↓	Throughput (example/s)↑	Latency (s)↓	Storage (MB)↓
Full PTM Fusion (FMF)	CodeBERT+BERT	0.793	0.808	0.864	0.870	4883	233	2.76	893
	CodeBERT+StarEncoder	0.810	0.818	0.858	0.864	5215	220	2.75	949
	CodeBERT+PLBART	0.728	0.752	0.856	0.862	5455	210	2.70	1007
	CodeBERT+CodeT5	0.706	0.741	0.850	0.860	7547	122	2.89	1321
	BERT+StarEncoder	0.774	0.793	0.862	0.868	4799	244	2.73	892
	BERT+PLBART	0.570	0.638	0.856	0.862	5141	200	2.65	949
	BERT+CodeT5	0.656	0.686	0.848	0.858	7259	134	2.75	1268
	StarEncoder+PLBART	0.660	0.715	0.858	0.865	5319	203	2.69	1005
	StarEncoder+CodeT5	0.649	0.686	0.861	0.869	7533	139	2.98	1324
	PLBART+CodeT5	0.665	0.705	0.841	0.854	7831	128	3.52	1382
Coding-Fuse	CodeBERT-6+BERT-5	*0.786*	**0.815**	*0.857*	*0.864*	**3207**	**393**	**1.94**	**542**
	CodeBERT-6+StarEncoder-7	*0.804*	*0.814*	**0.867**	**0.875**	**3725**	**393**	**2.14**	**652**
	CodeBERT-6+PLBART-5	**0.797**	**0.807**	**0.862**	**0.870**	**3359**	**328**	**2.07**	**601**
	CodeBERT-6+CodeT5-4	**0.785**	**0.797**	**0.852**	0.860	**3127**	**464**	**1.86**	**515**
	BERT-5+StarEncoder-6	**0.780**	**0.808**	**0.862**	**0.869**	**3103**	**398**	**1.87**	**540**
	BERT-5+PLBART-6	**0.773**	**0.784**	*0.854*	**0.864**	**3139**	**405**	**2.02**	**543**
	BERT-5+CodeT5-6	**0.799**	**0.814**	*0.845*	*0.853*	**2899**	**405**	**1.81**	**485**
	StarEncoder-6+PLBART-5	**0.748**	**0.784**	**0.860**	0.865	**3345**	**408**	**2.10**	**600**
	StarEncoder-6+CodeT5-4	**0.771**	**0.799**	**0.862**	**0.870**	**3109**	**431**	**1.85**	**514**
	PLBART-6+CodeT5-6	**0.783**	**0.792**	**0.850**	**0.859**	**3429**	**374**	**2.10**	**571**

TABLE VI

COMPARISON RESULTS OF MULTIPLE PERFORMANCE AND GREENNESS EVALUATION METRICS OF CODING-FUSE AND FMF ON THE CSD TASK IN CODE EMBEDDING SCENARIO AND FINETUNE MODEL SCENARIO. ↑ INDICATES THE LARGER THE BETTER.

Settings	Scenarios	Code embedding scenario		Finetune model scenario					
	Metrics	F1-Score↑	AUC-PR↑	F1-Score↑	AUC-PR↑	GPU (MB)↓	Throughput (example/s)↑	Latency (s)↓	Storage (MB)↓
Full PTM Fusion (FMF)	CodeBERT+BERT	0.602	0.687	0.509	0.620	9431	117	2.97	893
	CodeBERT+PLBART	0.596	0.688	0.505	0.613	11341	98	2.71	1007
	CodeBERT+CodeT5	0.570	0.674	0.467	0.584	17507	67	3.09	1321
	BERT+PLBART	0.577	0.679	0.539	0.643	11165	97	2.70	949
	BERT+CodeT5	0.568	0.668	0.574	0.668	17313	67	2.83	1268
	PLBART+CodeT5	0.548	0.654	0.468	0.587	19713	59	3.30	1382
Coding-Fuse	CodeBERT-7+BERT-6	**0.630**	**0.730**	**0.625**	**0.724**	**5807**	**209**	**2.01**	**596**
	CodeBERT-7+PLBART-6	**0.630**	**0.730**	**0.630**	**0.730**	**6893**	**181**	**2.22**	**655**
	CodeBERT-7+CodeT5-5	**0.596**	**0.698**	**0.517**	**0.624**	**6251**	**202**	**1.99**	**569**
	BERT-6+PLBART-6	**0.626**	**0.725**	**0.600**	**0.698**	**6405**	**195**	**2.14**	**570**
	BERT-6+CodeT5-5	**0.630**	**0.730**	**0.610**	**0.709**	**5761**	**222**	**1.88**	**485**
	PLBART-6+CodeT5-5	**0.630**	**0.730**	**0.537**	**0.641**	**7259**	**187**	**2.11**	**544**

the results of RQ1, the Coding-Fuse method further exploits the strength of different code PTMs than the FMF method and has better prospects.

Observation 4) The proposed efficient fusion code pre-trained model method Coding-Fuse has advantages over the full code pre-trained model fusion method FMF in terms of greenness, with higher efficiency and fewer hardware resources on the code-related classification tasks in the finetune model scenario, and at the same efficiency and hardware resource level as the corresponding original single PTM OSM. As shown in Tables IV to VI, in the fine-tuning model scenarios of the three classification tasks, the Coding-Fuse method has an overwhelming advantage in the four evaluation metrics of greenness compared to the FMF method, i.e., better in all cases. Specifically, for the CCD task, the GPU memory required to train all the parameters of the PTM fused by the Coding-Fuse method in the finetune model scenario is less than half of that of the FMF method, and the training throughput is generally improved by more than two times. Combined with the usage during deployment, we can find that compared with FMF, the Coding-Fuse method has a faster prediction speed, the test latency is reduced by 13% to 50%, and the model weight file only requires nearly half of the FMF hardware storage resources. For TDD and CSD tasks, the GPU memory required to train all parameters of the PTM fused by the Coding-Fuse method in the finetune model scenario is reduced by 29%-59% and 38%-67% compared with the FMF method, and the training throughput is increased by 56%-280% and 79%-231% respectively. The prediction latency during the final model test is reduced by 22%-40% and 18%-36% respectively, and the required storage resources are reduced by 31%-62% and 33%-62%. Combined with the original single pre-trained model (OSM) in RQ1 in the finetune model scenario, the Coding-Fuse method is at the same level

as OSM in terms of greenness evaluation. As shown in Table III (and more results can be found at https://github.com/ASE-2025 due to page limits), on the CCD task, the GPU memory and storage resources required to train all parameters of the PTM fused by the Coding-Fuse method are comparable to those of OSM, but it has certain advantages in terms of throughput during training and prediction latency during testing. Although the Coding-Fuse method has no particular advantage over OSM in hardware resources in TDD and CSD tasks, which is because the selected layers for these tasks are near the middle, leading to a larger merged model than OSM, it still achieves better values in terms of efficiency, i.e., training throughput and testing latency. In addition, in all the above experiments, the maximum GPU required for training Coding-Fuse does not exceed 27GB, and the maximum storage does not exceed 655MB, while the maximum GPU required for finetuning all OSMs requires 55GB and the maximum storage requires 850MB. Therefore, the Coding-Fuse method is still a better method that is competitive in terms of efficiency and resources on the whole.

Summary. Comprehensive performance and greenness, the Coding-Fuse can achieve better performance than FMF with higher efficiency and fewer hardware resources and can achieve better performance than OSM at the same efficiency and hardware resource level on the code-related classification tasks in both the code embedding scenario and the finetune model scenario.

VII. Discussion

In this section, we discuss the reasons for the effectiveness of Coding-Fuse and additional implications for SE practice, as well as the validity threats to conclusions.

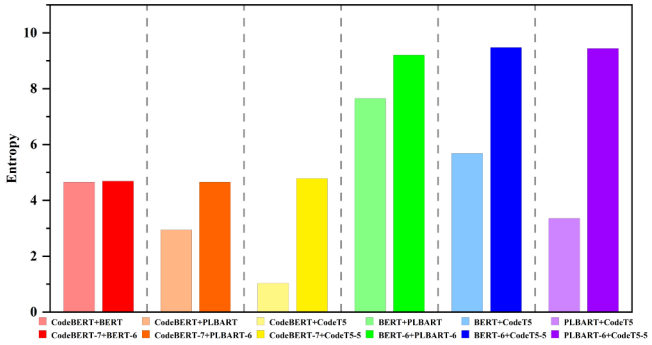

Fig. 3. Comparison of the average entropy of the output feature vectors between Coding-Fuse and FMF in the CSD task. The dark bar represents Coding-Fuse.

A. Why effective?

We hypothesize that such a result might be because the output features of the early layers of the fusion PTM provide richer information about the software code than that of the full PTM fusion. We arrive at such a hypothesis as prior studies [5], [55], [56] have shown that providing more information about the code helps improve the performance of SE classification tasks. In order to verify our hypothesis, we use entropy to quantify the amount of information in the output features of different methods. Specifically, the amount of information is compared by calculating the entropy value of the output feature vectors of all instances in the dataset [57], [58]. The larger the entropy value, the richer the information contained in the output feature vector. We show the comparison of the average entropy of the output feature vectors of Coding-Fuse and FMF on the code smell detection task in Figure 3 (more additional results can be found at https://github.com/ASE-2025).

The average entropy of the feature vectors output by Coding-Fuse in all cases is larger than that of FMF, which means that the informativeness of the feature vector output by Coding-Fuse is higher. This result supports our hypothesis that the Coding-Fuse output feature vector provides richer information, which in turn leads to the performance advantage of the classifier trained by the Coding-Fuse method over FMF.

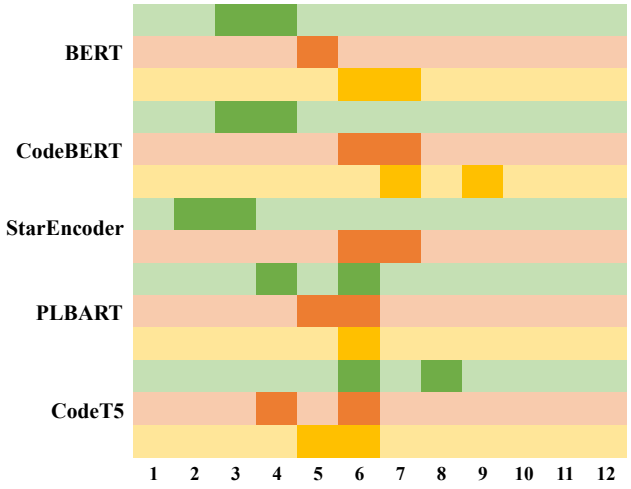

Fig. 4. The potential best layers found of Coding-Fuse, the green, orange and yellow colors represent CCD, TDD and CSD tasks respectively.

B. Implications for SE practitioner

SE researchers and practitioners could focus on only the early to middle layers when using code PTMs for downstream classification tasks. Figure 4 shows the potential best layers found in the first step of Coding-Fuse method, which can be found that the potential best layers of the five models on the three classification tasks are the early layers to the middle layers. The sixth middle layer is the most frequent potential best performance layer, which spreads to both sides, but the second to sixth left early layers are more frequent than the right sixth to ninth late layers. This conclusion is similar to the recent work of designing diverse probe tasks to explore and analyze code PTMs, e.g., Karmakar et.al [19], [20] pointed out that 9 of the 12 probe tasks performed better in the early-middle layers and middle layers of the code PTMs,

which proves the correctness of Coding-Fuse in locating the potential best performance layer from another perspective.

SE researchers and practitioners can apply the new paradigm of multi-pretrained model fusion to code classification tasks, and are recommended to try to use the proposed Coding-Fuse framework to generate fused new code pre-trained models. The code PTMs are injected with prior knowledge through the training process of the pre-training task. Based on this, the proposed Coding-Fuse can combine and splice the existing code PTMs to obtain brand new encoder and encoder-decoder code PTMs with different parameter scales and prior knowledge, inheriting the strength of multiple code PTMs instead of focusing on choosing a single code PTM or retraining a new code PTM. This provides a new paradigm to create new code PTMs that balance performance and greenness and meet personalized needs.

C. Threats to validity

Although the experiments in this study demonstrate the effectiveness of our method, the broader validity of these findings may still be subject to certain limitations.

Most of threats come from external validity. The conclusion for Coding-Fuse may be mostly valid on the code PTMs and the specific SE tasks we choose. We tried to minimize this threat by using five different code PTMs and three different SE classification tasks in two scenarios. However, due to hardware limitations, the model architectures and scales selected are still limited, and the experiment is just a fusion of two code PTMs. Therefore, it is still necessary to extend the experiments to more code PTMs and SE classification tasks to promote the generalizability of the conclusions. In addition, in our preliminary study, we only evaluated the FMF method on SE classification tasks with only one CCD task, which may limit the generalizability of preliminary study. But we believe the findings are likely to hold across other tasks, as PTM fusion enhances code understanding by providing richer information. The biggest threats to internal validity are the evaluation, while we counteract this threat by evaluating performance with F1-Score and AUC-PR and greenness by monitoring efficiency and resource with throughput and GPU memory occupied during training and the latency time during testing and the final storage size. A possible threat to construct validity is the hyperparameters, fusion method and classifier. While the optimal layer identification in Coding-Fuse would not be impacted by hyperparameters, since this process is hyperparameter-independent. For training the fused models involving hyperparameters (e.g., learning rate), we employ established optimal configurations from previous studies to conserve computational resources. In addition, we use the simplest soft voting fusion strategy and fully connected layer classifier for the fairness of the experiment, which means that these two may not fully exploit the advantages of different models. Nevertheless, Coding-Fuse still achieves considerable advantages in this case. However, more advanced fusion methods and classifiers should be used to fully exploit the advantages in the future.

VIII. Conclusion

Different background prior knowledge injected by different code PTMs during the pre-training process will lead to performance differences in downstream code-related classification tasks, which makes it a promising approach to improve task performance by fusing multiple code PTMs. This paper first proves this through a systematic exploratory study, but also finds that simply fusing two full code PTMs can only achieve limited performance gains and cannot unmet the greenness need. Therefore, we propose Coding-Fuse, a framework work of efficient fusion of code pre-trained models for classification tasks, which calculates the maximum evidence value to locate the layer with the potential best performance and fuses the layers of any two different code PTMs through a soft voting strategy to obtain a new intelligent model. Large-scale experiments on two task scenarios with five code PTMs and three SE classification tasks show that the proposed method has great advantages in performance and greenness. Our results provide clues for future SE researchers and practitioners to fast reuse code PTMs for intelligent model development and deployment to achieve performance, efficiency, and resource improvements.

Acknowledgements

This research is supported by the National Natural Science Foundation of China under grant No. U2241216 and No. 62202223, the Natural Science Foundation of Jiangsu Province under grant No. BK20220881, the Shenzhen Science and Technology Program under grant No. JCYJ20240813152507009, the Collaborative Innovation Center of Novel Software Technology and Industrialization and the High Performance Computing Platform of Nanjing University of Aeronautics and Astronautics.

References

[1] Z. Wan, X. Xia, A. E. Hassan *et al.*, "Perceptions, expectations, and challenges in defect prediction," *IEEE Trans. Software Eng.*, vol. 46, no. 11, pp. 1241–1266, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2018.2877678

[2] C. K. Roy and J. R. Cordy, "Benchmarks for software clone detection: A ten-year retrospective," in *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, R. Oliveto, M. D. Penta, and D. C. Shepherd, Eds. IEEE Computer Society, 2018, pp. 26–37. [Online]. Available: https://doi.org/10.1109/SANER.2018.8330194

[3] Z. Zeng, H. Tan, H. Zhang *et al.*, "An extensive study on pre-trained models for program understanding and generation," in *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, S. Ryu and Y. Smaragdakis, Eds. ACM, 2022, pp. 39–51. [Online]. Available: https://doi.org/10.1145/3533767.3534390

[4] C. Niu, C. Li, V. Ng *et al.*, "Comparing the pretrained models of source code by re-pretraining under a unified setup," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 35, no. 12, pp. 17 768–17 778, 2024. [Online]. Available: https://doi.org/10.1109/TNNLS.2023.3308595

[5] S. Jiang, Y. Chen, Z. He *et al.*, "Cross-project defect prediction via semantic and syntactic encoding," *Empir. Softw. Eng.*, vol. 29, no. 4, p. 80, 2024. [Online]. Available: https://doi.org/10.1007/s10664-024-10495-z

[6] Z. Yang, S. Chen, C. Gao *et al.*, "An empirical study of retrieval-augmented code generation: Challenges and opportunities," *CoRR*, vol. abs/2501.13742, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.13742

[7] Y. Zhao, L. Gong, Z. Huang *et al.*, "Coding-ptms: How to find optimal code pre-trained models for code embedding in vulnerability detection?" in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, V. Filkov, B. Ray, and M. Zhou, Eds. ACM, 2024, pp. 1732–1744. [Online]. Available: https://doi.org/10.1145/3691620.3695539

[8] Z. Bi, Y. Wan, Z. Chu *et al.*, "How to select pre-trained code models for reuse? A learning perspective," *CoRR*, vol. abs/2501.03783, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.03783

[9] Z. Sun, Q. Song, and X. Zhu, "Using coding-based ensemble learning to improve software defect prediction," *IEEE Trans. Syst. Man Cybern. Part C*, vol. 42, no. 6, pp. 1806–1817, 2012. [Online]. Available: https://doi.org/10.1109/TSMCC.2012.2226152

[10] N. Chen, Q. Sun, R. Zhu *et al.*, "Cat-probing: A metric-based approach to interpret how pre-trained models for programming language attend code structure," in *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Association for Computational Linguistics, 2022, pp. 4000–4008. [Online]. Available: https://doi.org/10.18653/v1/2022.findings-emnlp.295

[11] E. Shi, Y. Wang, H. Zhang *et al.*, "Towards efficient fine-tuning of pre-trained code models: An experimental study and beyond," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, R. Just and G. Fraser, Eds. ACM, 2023, pp. 39–51. [Online]. Available: https://doi.org/10.1145/3597926.3598036

[12] A. Grishina, M. Hort, and L. Moonen, "The earlybird catches the bug: On exploiting early layers of encoder models for more efficient code classification," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, S. Chandra, K. Blincoe, and P. Tonella, Eds. ACM, 2023, pp. 895–907. [Online]. Available: https://doi.org/10.1145/3611643.3616304

[13] G. A. Ahmed, J. V. Patten, Y. Han *et al.*, "Nearest-neighbor, bert-based, scalable clone detection: A practical approach for large-scale industrial code bases," *Softw. Pract. Exp.*, vol. 54, no. 12, pp. 2349–2374, 2024. [Online]. Available: https://doi.org/10.1002/spe.3355

[14] C. Ni, W. Wang, K. Yang *et al.*, "The best of both worlds: integrating semantic features with expert features for defect prediction and localization," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 672–683. [Online]. Available: https://doi.org/10.1145/3540250.3549165

[15] A. Kovacevic, N. Luburic, J. Slivka *et al.*, "Automatic detection of code smells using metrics and codet5 embeddings: a case study in c#," *Neural Comput. Appl.*, vol. 36, no. 16, pp. 9203–9220, 2024. [Online]. Available: https://doi.org/10.1007/s00521-024-09551-y

[16] W. Ma, Y. Yu, X. Ruan *et al.*, "Pre-trained model based feature envy detection," in *20th IEEE/ACM International Conference on Mining Software Repositories, MSR 2023, Melbourne, Australia, May 15-16, 2023*. IEEE, 2023, pp. 430–440. [Online]. Available: https://doi.org/10.1109/MSR59073.2023.00065

[17] Z. Zeng, Y. Zhao, and L. Gong, "Classifying bug issue types for deep learning-oriented projects with pre-trained model," in *31st Asia-Pacific Software Engineering Conference, APSEC 2024, Chongqing, China, December 3-6, 2024*. IEEE, 2024, pp. 91–100. [Online]. Available: https://doi.org/10.1109/APSEC65559.2024.00020

[18] V.-H. Le, H. Zhang, and Y. Xiao, " Unleashing the True Potential of Semantic-based Log Parsing with Pre-trained Language Models ," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 711–711. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00174

[19] A. Karmakar and R. Robbes, "What do pre-trained code models know about code?" in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 1332–1336. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678927

[20] ——, "INSPECT: intrinsic and systematic probing evaluation for code transformers," *IEEE Trans. Software Eng.*, vol. 50, no. 2, pp. 220–238, 2024. [Online]. Available: https://doi.org/10.1109/TSE.2023.3341624

[21] W. Ma, S. Liu, M. Zhao *et al.*, "Unveiling code pre-trained models: Investigating syntax and semantics capacities," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 7, pp. 169:1–169:29, 2024. [Online]. Available: https://doi.org/10.1145/3664606

[22] Y. Wan, W. Zhao, H. Zhang *et al.*, "What do they capture? - A structural analysis of pre-trained language models for source code," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 2377–2388. [Online]. Available: https://doi.org/10.1145/3510003.3510050

[23] J. Devlin, M. Chang, K. Lee *et al.*, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[24] Z. Feng, D. Guo, D. Tang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1536–1547. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.139

[25] R. Li, L. B. Allal, Y. Zi *et al.*, "Starcoder: may the source be with you!" *Trans. Mach. Learn. Res.*, vol. 2023, 2023. [Online]. Available: https://openreview.net/forum?id=KoFOg41haE

[26] W. U. Ahmad, S. Chakraborty, B. Ray *et al.*, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, K. Toutanova, A. Rumshisky, L. Zettlemoyer *et al.*, Eds. Association for Computational Linguistics, 2021, pp. 2655–2668. [Online]. Available: https://doi.org/10.18653/v1/2021.naacl-main.211

[27] Y. Wang, W. Wang, S. R. Joty *et al.*, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, M. Moens, X. Huang, L. Specia *et al.*, Eds. Association for Computational Linguistics, 2021, pp. 8696–8708. [Online]. Available: https://doi.org/10.18653/v1/2021.emnlp-main.685

[28] C. Niu, C. Li, V. Ng *et al.*, "An empirical comparison of pre-trained models of source code," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 2136–2148. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00180

[29] M. Khajezade, J. J. Wu, F. H. Fard *et al.*, "Investigating the efficacy of large language models for code clone detection," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, ICPC 2024, Lisbon, Portugal, April 15-16, 2024*, I. Steinmacher, M. Linares-Vásquez, K. P. Moran *et al.*, Eds. ACM, 2024, pp. 161–165. [Online]. Available: https://doi.org/10.1145/3643916.3645030

[30] F. Zhang, Z. Zhang, J. W. Keung *et al.*, "Data preparation for deep learning based code smell detection: A systematic literature review," *J. Syst. Softw.*, vol. 216, p. 112131, 2024. [Online]. Available: https://doi.org/10.1016/j.jss.2024.112131

[31] J. P. Biazotto, D. Feitosa, P. Avgeriou *et al.*, "Technical debt management automation: State of the art and future perspectives," *Inf. Softw. Technol.*, vol. 167, p. 107375, 2024. [Online]. Available: https://doi.org/10.1016/j.infsof.2023.107375

[32] J. Svajlenko, J. F. Islam, I. Keivanloo *et al.*, "Towards a big data curated benchmark of inter-project code clones," in *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE Computer Society, 2014, pp. 476–480. [Online]. Available: https://doi.org/10.1109/ICSME.2014.77

[33] W. Wang, G. Li, B. Ma *et al.*, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou

et al., Eds. IEEE, 2020, pp. 261–271. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054857

[34] S. Fakhoury, V. Arnaoudova, C. Noiseux *et al.*, "Keep it simple: Is deep learning good for linguistic smell detection?" in *25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018*, R. Oliveto, M. D. Penta, and D. C. Shepherd, Eds. IEEE Computer Society, 2018, pp. 602–611. [Online]. Available: https://doi.org/10.1109/SANER.2018.8330265

[35] V. Arnaoudova, M. D. Penta, and G. Antoniol, "Linguistic antipatterns: what they are and how developers perceive them," *Empir. Softw. Eng.*, vol. 21, no. 1, pp. 104–158, 2016. [Online]. Available: https://doi.org/10.1007/s10664-014-9350-8

[36] E. da S. Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt," *IEEE Trans. Software Eng.*, vol. 43, no. 11, pp. 1044–1062, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2017.2654244

[37] K. Liu, G. Yang, X. Chen *et al.*, "El-codebert: Better exploiting codebert to support source code-related classification tasks," in *Internetware 2022: 13th Asia-Pacific Symposium on Internetware, Hohhot, China, June 11 - 12, 2022*. ACM, 2022, pp. 147–155. [Online]. Available: https://doi.org/10.1145/3545258.3545260

[38] S. Lu, D. Guo, S. Ren *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, J. Vanschoren and S. Yeung, Eds., 2021. [Online]. Available: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c16a5320fa475530d9583c34fd356ef5-Abstract-round1.html

[39] Z. Ding, H. Li, W. Shang *et al.*, "Can pre-trained code embeddings improve model performance? revisiting the use of code embeddings in software engineering tasks," *Empir. Softw. Eng.*, vol. 27, no. 3, p. 63, 2022. [Online]. Available: https://doi.org/10.1007/s10664-022-10118-5

[40] X. Zhou, D. Han, and D. Lo, "Assessing generalizability of codebert," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021*. IEEE, 2021, pp. 425–436. [Online]. Available: https://doi.org/10.1109/ICSME52107.2021.00044

[41] A. F. del Carpio and L. B. Angarita, "Trends in software engineering processes using deep learning: A systematic literature review," in *46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, Portoroz, Slovenia, August 26-28, 2020*. IEEE, 2020, pp. 445–454. [Online]. Available: https://doi.org/10.1109/SEAA51224.2020.00077

[42] Y. Yang, X. Xia, D. Lo *et al.*, "A survey on deep learning for software engineering," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 206:1–206:73, 2022. [Online]. Available: https://doi.org/10.1145/3505243

[43] D. Gray, D. Bowes, N. Davey *et al.*, "Further thoughts on precision," in *15th International Conference on Evaluation & Assessment in Software Engineering, EASE 2011, Durham, UK, 11-12 April 2011, Proceedings*. IET - The Institute of Engineering and Technology / IEEE Xplore, 2011, pp. 129–133. [Online]. Available: https://doi.org/10.1049/ic.2011.0016

[44] J. Davis and M. H. Goadrich, "The relationship between precision-recall and ROC curves," in *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, ser. ACM International Conference Proceeding Series, W. W. Cohen and A. W. Moore, Eds., vol. 148. ACM, 2006, pp. 233–240. [Online]. Available: https://doi.org/10.1145/1143844.1143874

[45] Y. Tang, Y. Zhang, N. V. Chawla *et al.*, "Svms modeling for highly imbalanced classification," *IEEE Trans. Syst. Man Cybern. Part B*, vol. 39, no. 1, pp. 281–288, 2009. [Online]. Available: https://doi.org/10.1109/TSMCB.2008.2002909

[46] J. Shi, Z. Yang, and D. Lo, "Efficient and green large language models for software engineering: Vision and the road ahead," *ACM Trans. Softw. Eng. Methodol.*, Dec. 2024, just Accepted. [Online]. Available: https://doi.org/10.1145/3708525

[47] G. Yang, Y. Zhou, X. Zhang *et al.*, "Less is more: Towards green code large language models via unified structural pruning," *CoRR*, vol. abs/2412.15921, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.15921

[48] A. Nagarajan and A. Raghunathan, "Tokendrop + bucketsampler: Towards efficient padding-free fine-tuning of language models," in *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Association for Computational Linguistics, 2023, pp. 11 682–11 695. [Online]. Available: https://doi.org/10.18653/v1/2023.findings-emnlp.782

[49] Z. Lin, Z. Gou, Y. Gong *et al.*, "Not all tokens are what you need for pretraining," in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave *et al.*, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/3322a9a72a1707de14badd5e552ff466-Abstract-Conference.html

[50] M. Toker, I. Galil, H. Orgad *et al.*, "Padding tone: A mechanistic analysis of padding tokens in T2I models," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, L. Chiruzzo, A. Ritter, and L. Wang, Eds. Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 7618–7632. [Online]. Available: https://aclanthology.org/2025.naacl-long.389/

[51] K. H. Knuth, M. Habeck, N. K. Malakar *et al.*, "Bayesian evidence and model selection," *Digit. Signal Process.*, vol. 47, pp. 50–67, 2015. [Online]. Available: https://doi.org/10.1016/j.dsp.2015.06.012

[52] K. You, Y. Liu, J. Wang *et al.*, "Logme: Practical assessment of pre-trained models for transfer learning," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 12 133–12 143. [Online]. Available: http://proceedings.mlr.press/v139/you21b.html

[53] S. F. Gull, "Developments in maximum entropy data analysis," in *Maximum Entropy and Bayesian Methods: Cambridge, England, 1988*. Springer, 1989, pp. 53–71.

[54] Y. Chai, H. Zhang, B. Shen *et al.*, "Cross-domain deep code search with meta learning," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 487–498. [Online]. Available: https://doi.org/10.1145/3510003.3510125

[55] Y. Wu, S. Feng, D. Zou *et al.*, "Detecting semantic code clones by building ast-based markov chains model," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 34:1–34:13. [Online]. Available: https://doi.org/10.1145/3551349.3560426

[56] D. Wu, F. Mu, L. Shi *et al.*, "ismell: Assembling llms with expert toolsets for code smell detection and refactoring," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE 2024, Sacramento, CA, USA, October 27 - November 1, 2024*, V. Filkov, B. Ray, and M. Zhou, Eds. ACM, 2024, pp. 1345–1357. [Online]. Available: https://doi.org/10.1145/3691620.3695508

[57] S. K. Abd-El-Hafiz, "Entropies as measures of software information," in *2001 International Conference on Software Maintenance, ICSM 2001, Florence, Italy, November 6-10, 2001*. IEEE Computer Society, 2001, pp. 110–117. [Online]. Available: https://doi.org/10.1109/ICSM.2001.972721

[58] M. Kumari, R. Singh, and V. B. Singh, "Prioritization of software bugs using entropy-based measures," *J. Softw. Evol. Process.*, vol. 37, no. 2, 2025. [Online]. Available: https://doi.org/10.1002/smr.2742