Empirical Software Engineering (2026) 31:151
https://doi.org/10.1007/s10664-026-10880-w

Exploring and improving knowledge distillation for pre-trained code models

Weifeng Sun[1] · Ruifeng Wu[1] · Hongyan Li[1] · Ying Fu[3] · Min Yu[2] · Meng Yan[1]

Received: 17 December 2025 / Accepted: 11 May 2026

Abstract
Recently, pre-trained code models (PCMs) have advanced software development by automating tasks and improving productivity. However, their large size hinders seamless adoption in developers' daily workflows, as local execution is often infeasible and reliance on cloud-based services raises concerns about data privacy. These challenges highlight the need for effective compression techniques to enable secure and efficient on-device deployment. To address this problem, this paper investigates knowledge distillation (KD) as a means to compress large PCMs. We systematically evaluate the effectiveness of KD on PCMs by comparing different distillation paradigms across various code generation and understanding tasks. Our results show that feature-based distillation generally outperforms response-based approaches, though the effectiveness varies depending on the specific PCM and downstream task. Furthermore, we identify key factors that influence the effectiveness of the feature-based knowledge distillation, including loss functions, mapping configurations, and the number of intermediate layer distillation iterations. Building on these insights, we propose BOKD, a Bayesian optimization–based method that adaptively selects optimal distillation configurations according to the student model's capacity and task complexity. Empirical results demonstrate the practicality of BOKD. A compact 2-layer model (just 5% of teacher parameters) retains 96–97% (relative performance score) of teacher performance on classification tasks, while achieving a 30% perplexity reduction over baseline distillation methods. Our contributions include a comprehensive empirical studies on distillation paradigms, and a novel parameter optimization technique to enhance KD performance.

Keywords Knowledge distillation · Pre-trained code models · Model compression · Configuration tuning

Weifeng Sun and Ruifeng Wu contributed equally to this work.

Communicated by: Cuiyin Gao, Kui Liu, Xin Xia and David Lo.

Extended author information available on the last page of the article

Published online: 23 May 2026

1 Introduction

In recent years, Artificial Intelligence (AI)-powered services have revolutionized software engineering by introducing advanced automation into the development process. The emergence of numerous programming assistance tools, such as GitHub Copilot[1], Amazon Q[2], and GitLab Auto DevOps[3], has reshaped traditional coding workflows. This transformation has significantly boosted developer efficiency and improved the overall quality of software products (Niu et al. 2022, 2023a; Zeng et al. 2022). According to an economic analysis report released by GitHub, AI-powered services for software development can boost the global GDP by over $1.5 trillion by 2030 (Dohmke et al. 2023).

The foundation of many AI-driven software engineering services lies in PCMs, such as CodeBERT (Feng et al. 2020) and CodeT5+ (Wang et al. 2023). However, current deployments of PCM-based services are predominantly cloud-based, which raises concerns about potential data leakage (Huang et al. 2023; Lo 2023; Niu et al. 2023b; Yang et al. 2023) and leads to degraded user experience due to network latency (GitHub Copilot Community 2023). As a result, there is an increasing demand for deploying these models directly within integrated development environments (IDEs) on developers' local machines. Recent studies (Wei et al. 2023; Shi et al. 2022) have underscored the obstacles to such deployment, including the large parameter size of PCMs and their considerable inference latency.

Typically, PCMs are characterized by a large number of parameters. For instance, CodeBERT (Feng et al. 2020) and GraphCodeBERT (Guo et al. 2020), widely-used PCMs, each have 125 million parameters, resulting in file sizes of approximately 500 megabytes (MB). Their considerable size makes it difficult to deploy them on edge devices like personal computers and laptops (Svyatkovskiy et al. 2021). In addition, these models require considerable computational resources that are often unavailable on local machines, leading to inference delays exceeding 1.5 seconds per prediction (Shi et al. 2022). Consequently, model compression has emerged as a critical research direction, aiming to reduce the storage and computational overhead of PCMs while preserving their performance under stringent hardware constraints (Tang et al. 2019).

To date, various approaches have been proposed to compress pre-trained models in natural language processing or other tasks (Zhu et al. 2024; Han et al. 2015a; Jiao et al. 2020; Sun et al. 2020; Zhang et al. 2020; Xu et al. 2020). These existing studies fall under three types: model pruning, model quantization, and knowledge distillation. Model pruning reduces model size by setting a portion of parameters to zero or removing architectural components such as layers or attention heads (Han et al. 2015b). However, pruning is often limited by the inherent structure of the model, making it challenging to shrink models to a size suitable for deployment on resource-constrained devices. For example, even if all network layers of a pre-trained model such as CodeBERT are pruned, the embedding table alone still occupies approximately 150 MB (Shi et al. 2022). Model quantization compresses models by converting parameters from 32-bit floating-point representations to lower-bit fixed-point numbers (Gong et al. 2014). While quantization significantly reduces model size, it often fails to improve inference speed or reduce CPU memory usage unless specialized

[1] https://github.com/features/copilot (accessed: 2025.08.22)

[2] https://aws.amazon.com/cn/q/developer/?nc1=h_ls. (accessed: 2025.08.22)

[3] https://about.gitlab.com/blog/2022/02/14/top-10-ways-machine-learning-may-help-devops/ (accessed: 2025.08.22)

hardware or optimized processing libraries are available (Ganesh et al. 2021; Kanade et al. 2020; Zadeh et al. 2020). In contrast, knowledge distillation (KD) offers a more promising pathway for model compression (Hinton 2015; Romero et al. 2014). KD adopts a teacher–student paradigm in which a large teacher network transfers its knowledge to a smaller student network. The student is trained to approximate the behavior of the teacher, yielding a compressed model that not only achieves a substantial reduction in size but also delivers efficiency gains on commodity CPUs.

Within the Knowledge Distillation (KD) framework, most prior studies have concentrated on models and tasks in natural language processing (NLP) and computer vision (CV) (Zhu et al. 2024; Gou et al. 2021; Phuong and Lampert 2019), with comparatively little attention given to pre-trained code models (PCMs). Although a few works have explored KD in the context of PCMs, these efforts typically adopt KD in a straightforward manner to reduce model size (Shi et al. 2022, 2023). This simplicity presents a critical limitation. Unlike natural language, code is highly structured, syntax-dependent, and governed by strict semantic rules. Consequently, the crucial knowledge learned by large teacher PCMs—such as understanding syntactic dependencies, abstract syntax trees (ASTs), and data flow—is fundamentally different from the unstructured or sequential knowledge captured in general NLP models (Karmakar and Robbes 2021). We hypothesize that applying off-the-shelf KD strategies, which were designed primarily for NLP data, is inherently suboptimal for transferring this structural code knowledge. Therefore, a substantial and necessary gap remains in understanding how to effectively apply and optimally configure KD paradigms specifically for PCMs (Xu et al. 2022). Our work addresses this gap through a systematic investigation into distillation parameters to unlock significant performance gains in compact student code models.

In this paper, we investigate the role of KD in compressing PCMs. We first review established KD techniques and summarize two widely adopted paradigms in language-related tasks: response-based distillation and feature-based distillation (Xu et al. 2024; Gou et al. 2021). By conducting evaluations across three downstream tasks and two student model sizes, we analyze the impact of these paradigms on large PCMs with different architectures. Our empirical results indicate that the relative effectiveness of each paradigm varies depending on the task and architecture, with feature-based distillation generally achieving stronger performance than response-based distillation. Building on this, we provide a deeper analysis of the feature-based KD process by investigating key factors that influence its success. Specifically, we examine how the choice of loss function for computing attention loss, the design of the feature-mapping function, and the number of intermediate-layer distillation iterations affect distillation outcomes. Our findings highlight that these parameters play a critical role, and no single configuration consistently outperforms others across all scenarios. Instead, optimal parameter settings must be carefully adapted to the characteristics of both the downstream task and the student model.

To address this challenge, we propose BOKD (Bayesianly Optimized Intermediate-layer Knowledge Distillation), a feature-based KD framework enhanced by automated parameter optimization. BOKD employs Bayesian optimization to efficiently explore the design space of key distillation parameters including loss functions, feature-mapping strategies, and the number of intermediate-layer iterations and to select strategies tailored to the capacity of the student model and the complexity of the downstream task. Experimental results across three student model sizes demonstrate that BOKD achieves consistently superior perfor-

mance, providing an adaptive and effective solution for compressing PCMs. For example, our 6-layer student model, with only 56% of the teacher's parameters, achieves near-teacher accuracy on vulnerability detection with less than 1% loss. Even the most compact 2-layer model, using merely 5% of the parameters, retains over 95% of teacher performance across tasks. These results demonstrate the effectiveness of BOKD in producing lightweight yet competitive models, enabling practical deployment in resource-constrained environments.

The contributions of this paper are summarized as follows:

- **Empirical Study.** We systematically evaluate knowledge distillation (KD) paradigms on three representative downstream tasks (vulnerability detection, code clone detection, and code documentation generation) and across varying student sizes, showing that feature-based KD generally outperforms response-based KD, but its effectiveness is highly configuration- and architecture-dependent.
- **Insight.** We reveal that KD effectiveness in pre-trained code models (PCMs) is architecture-dependent. Different PCM types (encoder-only, decoder-only, encoder–decoder) exhibit distinct sensitivities to layer mapping, loss design, and distillation depth, revealing unique trade-offs in code knowledge transfer absent in general NLP KD studies.
- **Technical Contribution.** We propose BOKD (**B**ayesianly **O**ptimized Intermediate-layer **K**nowledge **D**istillation), a framework that couples Bayesian optimization with feature-based KD to automatically select task- and model-aware configurations. BOKD enables compact student models to retain near-teacher performance under strict resource constraints.
- **Open-source.** We release all code, experimental scripts, and distilled models at https://anonymous.4open.science/r/KDcode2public-099C/ to facilitate future research and practical adoption.

2 Background

2.1 Pre-trained Code Model

Pre-trained models for natural languages, such as BERT (Devlin et al. 2018) and Qwen (Yang et al. 2025) have achieved remarkable improvements across a wide range of language tasks in recent years. Their success largely stems from training deep neural networks on massive text corpora using self-supervised learning objectives. Motivated by this success, the software engineering (SE) community has developed a variety of PCMs (Lin et al. 2023; Feng et al. 2020; Guo et al. 2020; Hui et al. 2024), which leverage large-scale programming language corpora and have shown strong performance on diverse code understanding and generation tasks. PCMs can be broadly categorized into three types (Zhang et al. 2023): (1) *encoder-only models*, which employ only the encoder component, such as CodeBERT (Feng et al. 2020) and GraphCodeBERT (Guo et al. 2020; 2) *decoder-only models*, which adopt only the decoder component, such as Qwen2.5-Coder (Hui et al. 2024) and CodeGen (Nijkamp et al. 2022); and (3) *encoder–decoder models*, which integrate both components, such as CodeT5+ (Wang et al. 2023) and T5-code (Mastropaolo et al. 2021).

2.2 Knowledge Distillation

As data and model sizes continue to grow, deep neural networks (DNNs) have achieved remarkable performance across diverse tasks. However, their large parameter counts and the limited computational resources of deployment environments pose significant challenges. To address this issue, knowledge distillation (KD) has emerged as a particularly effective model compression technique (Gou et al. 2021). KD compresses a large model by training a smaller *student* model to reproduce the behavior of a larger *teacher* model (Ba and Caruana 2014; Gou et al. 2021; Hinton 2015). Through this process, the student learns to produce outputs consistent with those of the teacher for identical inputs, thereby yielding a compact model that achieves competitive accuracy with substantially reduced computational overhead. Extensive research has validated the effectiveness of KD across diverse domains (Yoon et al. 2021; Liu et al. 2018; Cho and Hariharan 2019).

Existing KD paradigms can be broadly classified into three categories based on the form of knowledge transferred: *response-based*, *feature-based*, and *relation-based* distillation (Gou et al. 2021). While relation-based distillation captures structural relationships among samples and excels in computer vision tasks, response-based and feature-based methods are more suitable for natural language processing tasks where token-level and sequential information dominate. Accordingly, this study focuses on response-based and feature-based paradigms to evaluate KD effectiveness for PCMs.

Response-based distillation transfers knowledge by aligning the final output distributions between teacher and student. It directly minimizes the discrepancy between their output logits, typically using cross-entropy loss (Mirzadeh et al. 2020; Hinton 2015). Ground-truth labels are sometimes incorporated to provide additional supervision. Despite its simplicity, response-based distillation has proven effective across a wide range of tasks.

In contrast, *feature-based distillation* leverages the teacher's intermediate representations, such as hidden states and feature maps, as additional supervision signals. This approach enables the student to capture richer internal knowledge beyond final predictions (Gou et al. 2021; Xu et al. 2024). State-of-the-art feature-based frameworks typically employ a two-stage process: The first stage transfers knowledge from intermediate layers, including embeddings and Transformer blocks; the second stage distills knowledge from the final output logits, referred to as *prediction-layer distillation* (Jiao et al. 2020; Sanh et al. 2019). Figure 1 illustrates the workflow of these two paradigms, highlighting their key differences.

Although numerous PCMs have been proposed, prior work (Lu et al. 2021) indicates that performance differences among models sharing the same architecture are typically much smaller than those observed across different architectures. This finding highlights the importance of investigating knowledge distillation techniques on PCMs with heterogeneous architectures. Accordingly, in this study we focus on strong and representative models drawn from each of the three architecture families.

Our study adopts an *offline distillation* setting to examine KD for compressing PCMs on specific downstream tasks. In this setup, a large pre-trained code model fine-tuned for the target task serves as the teacher, while a smaller student model is initialized without task-specific training (Jiao et al. 2020; Xu et al. 2024). Training data is passed through both models to obtain the teacher's output probabilities and intermediate features, which serve as supervisory signals for the student. By minimizing the corresponding distillation loss, the

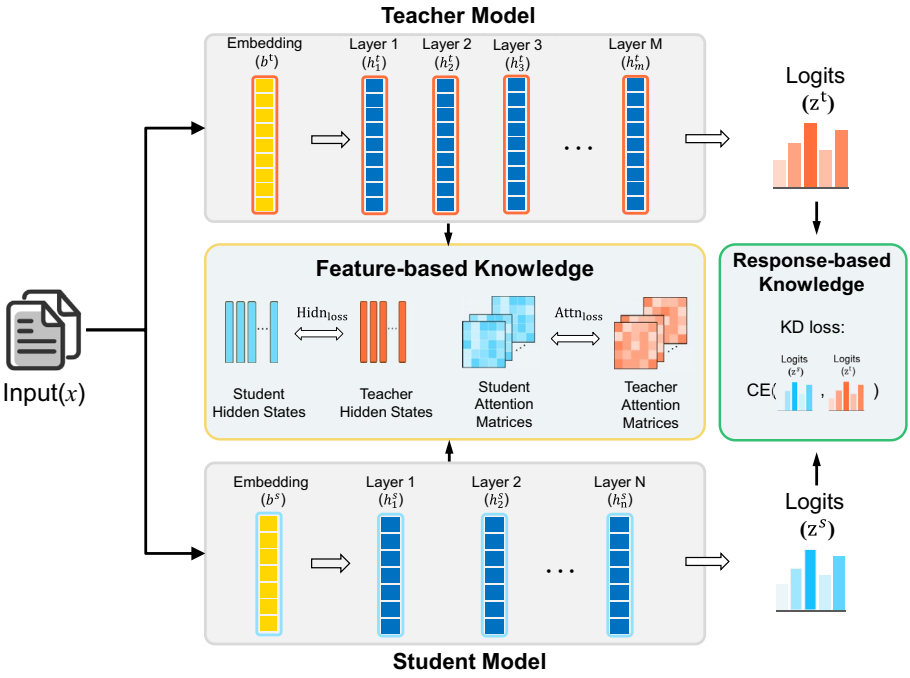

Fig. 1 Overview of knowledge distillation frameworks. The figure illustrates two primary knowledge distillation paradigms

student's predictions and internal representations are progressively aligned with those of the teacher (Gou et al. 2021).

3 Experimental Setup for Knowledge Distillation in Pre-trained Code Models

3.1 Distillation Workflow

Figure 2 illustrates the basic workflow of *offline knowledge* distillation from PCMs, which consists of two main stages: (i) *teacher model fine-tuning* and (ii) *knowledge distillation*, detailed as follows. **Teacher Model Fine-Tuning Phase.** A pre-trained code model is selected as the teacher, typically consisting of a deep stack of encoder and/or decoder layers that capture rich code semantics. The teacher model is then fine-tuned on task-specific data to adapt it to the target downstream task. This procedure produces a high-capacity teacher model that leverages knowledge from large-scale pre-training while being optimized to capture task-specific nuances essential for effective performance. **Knowledge Distillation Phase.** After fine-tuning, a smaller student model is introduced, designed to satisfy hardware constraints and reduce inference latency, thereby enhancing the practicality of PCMs in personal computing environments. The student model undergoes learning under the supervision of the teacher, aligning its predictions and representations with those of the teacher. This process transfers the knowledge encoded in the teacher into a more com-

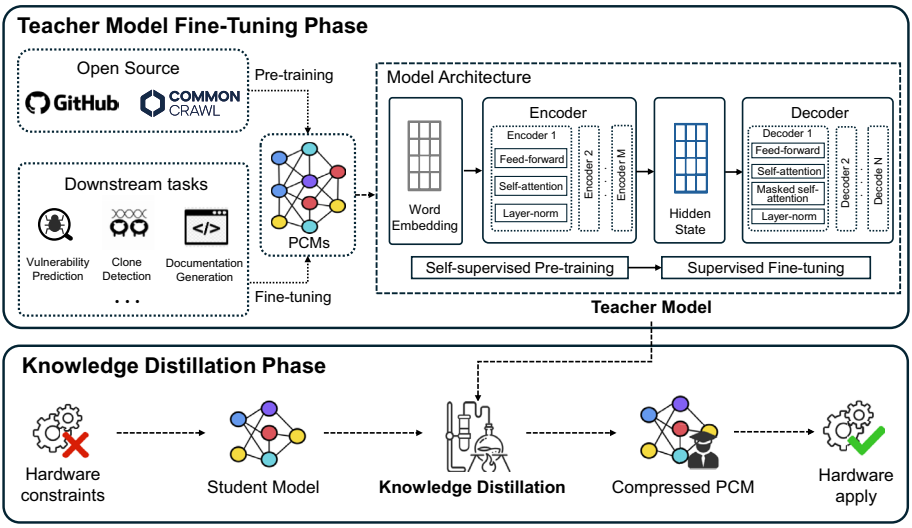

Fig. 2 An overview workflow of distilling knowledge from PCMs

pact yet effective student model, achieving a balance between model efficiency and task performance.

3.2 Research Questions

This study is guided by the following three research questions (RQs):

RQ1: How do different knowledge distillation paradigms affect the performance of compressed PCMs on downstream tasks? We systematically evaluate and compare mainstream knowledge distillation paradigms applied to PCMs with diverse architectures across multiple downstream tasks. The goal is to understand how these paradigms influence the performance characteristics of distilled models.

RQ2: What is the impact of parameter configurations on the effectiveness of knowledge distillation in PCMs? Building on RQ1, we investigate the role of parameter configurations in the distillation process. This analysis aims to identify which configurations critically affect the effectiveness of knowledge distillation and to quantify their impact through empirical evaluation.

RQ3: Can the effectiveness of existing knowledge distillation techniques be further improved through efficient parameter optimization? Motivated by the findings of RQ2, we explore whether automated and efficient parameter optimization strategies can further enhance the effectiveness of knowledge distillation in PCMs.

Overall, this study provides a comprehensive evaluation of knowledge distillation for PCMs by analyzing distillation paradigms, examining the influence of parameter configurations, and proposing optimization approaches that improve the effectiveness of KD on downstream tasks.

3.3 Model Selection

In our experiments, we select three distinct teacher models corresponding to the three primary Transformer architectures: encoder-only, encoder-decoder, and decoder-only.

- **CodeBERT:** CodeBERT (Feng et al. 2020) is an encoder-only, bi-modal large language model (LLM) pre-trained to understand both programming and natural languages. Its training objective combines masked language modeling (MLM) and replaced token detection: MLM predicts original tokens that have been masked, while replaced token detection identifies whether a given token is original or has been substituted. Notably, CodeBERT has demonstrated strong performance across a variety of code-related tasks in the CodeXGLUE benchmark (Lu et al. 2021), a widely used platform for evaluating machine learning models on program understanding and generation.
- **CodeT5+:** For the encoder-decoder architecture, we adopt CodeT5+ (Wang et al. 2023) as the teacher model. CodeT5+ is a unified pre-trained encoder-decoder model specifically designed for code-related tasks. It extends the original CodeT5 (Wang et al. 2021) by incorporating a larger pre-training corpus, more diverse programming languages, and an improved training objective that combines span denoising and identifier-aware pre-training strategies. These enhancements enable CodeT5+ to better capture both syntactic structures and semantic relationships in source code, making it highly effective for tasks such as code summarization, translation, and generation.
- **Qwen2.5-Coder:** For the decoder-only architecture, we employ Qwen2.5-Coder (Hui et al. 2024) as the teacher model. Qwen2.5-Coder is a state-of-the-art large-scale autoregressive language model optimized for code understanding and generation. Built upon the decoder-only Transformer architecture, it is pre-trained on a diverse and extensive code corpus covering multiple programming languages and domains. Qwen2.5-Coder incorporates advanced training techniques such as instruction tuning and multi-task learning, which significantly improve its ability to follow task-specific prompts and generate high-quality code (Hui et al. 2024).

In alignment with the architecture of each selected teacher model, we design a set of student models that preserve the same structural paradigm but differ in scale. For each teacher spanning encoder-only, encoder-decoder, and decoder-only Transformer types we construct three corresponding student models of decreasing size. These models share the same architectural characteristics as their respective teachers but are reduced in terms of layer depth and hidden size to accommodate lightweight deployment. Table 1 presents the full architectural details of all teacher and student models used in our experiments.

3.4 Downstream Tasks

To comprehensively explore the effectiveness of knowledge distillation on PCMs, we consider three important downstream tasks: vulnerability prediction, clone detection, and code documentation generation.

- **Vulnerability Prediction:** This task focuses on predicting whether a given code snippet is vulnerable, with the goal of integrating such a model into IDEs to help developers

Table 1 Model architecture configurations of teacher and student models. Notation: L_{enc} – encoder layers, L_{dec} – decoder layers, H – hidden size, I – intermediate size, A – number of attention heads

Model	L_{enc}	L_{dec}	H	I	A	Size
Teacher PCMs						
CodeBERT	12	0	768	3072	12	126M
CodeT5+	12	12	768	3072	12	220M
Qwen2.5-Coder	0	24	896	4864	14	494M
CodeBERT Student Models						
CodeBERT-6layer	6	0	768	3072	12	82.7M
CodeBERT-4layer	4	0	312	1200	12	20.7M
CodeBERT-2layer	2	0	144	450	12	7.8M
CodeT5+ Student Models						
CodeT5+-6layer	6	6	768	3072	12	124M
CodeT5+-4layer	4	4	312	1200	12	27.8M
CodeT5+-2layer	2	2	144	450	12	7.9M
Qwen2.5-Coder Student Models						
Qwen2.5-Coder-6layer	0	6	896	4864	14	226M
Qwen2.5-Coder-4layer	0	4	336	1200	14	56.9M
Qwen2.5-Coder-2layer	0	2	168	450	14	26.3M

detect potential code defects early. Recent advances in this field have demonstrated state-of-the-art performance through various deep learning approaches, including transformer-based architectures and graph neural networks (Wang et al. 2024; Al Debeyan et al. 2025). Specifically, we utilize the dataset validated in the Avatar study (Shi et al. 2023), which is derived from the Devign dataset (Zhou et al. 2019). Devign contains 27,318 functions collected from two widely used open-source C libraries, FFmpeg and Qemu, with each function manually labeled as either vulnerable or non-vulnerable. To ensure the correctness and comparability of our experiments, we follow the data split protocol from CodeXGLUE (Lu et al. 2021), using 2,732 examples for validation and testing, and the remaining examples for training.

– **Clone Detection:** This task focuses on determining whether two given functions are clones, meaning they share equivalent operational semantics. Clone detection is crucial for identifying redundant implementations and helping developers enhance software quality through code refactoring, remaining an active research area with continued state-of-the-art advancements (Yang et al. 2024; Feng et al. 2024). Similar to the Vulnerability Prediction task, we utilize the dataset provided in the Avatar study (Shi et al. 2023), which is randomly sampled from BigCloneBench (Svajlenko et al. 2014), a widely used benchmark for clone detection in open-source Java projects. BigCloneBench contains over 6 million cloned method pairs and 260,000 non-clone pairs. To balance computational efficiency and experimental reliability, the Avatar study adopts a subset consisting of 90,102 examples for training and 4,000 examples for validation and testing.
– **Code Documentation Generation:** This task focuses on generating natural language documentation for given code snippets, aiming to assist developers in understanding and maintaining code more efficiently. With the rapid advancement of generative AI, automatic code documentation generation has become a cutting-edge research frontier (Luo et al. 2024; Dvivedi et al. 2024). For this study, we evaluate the performance of our model on the documentation generation task using the CodeSearchNet Corpus (Husain

et al. 2019). The CodeSearchNet dataset provides a structured and diverse collection of data, including metadata such as function names, docstrings, and code contexts, enabling comprehensive evaluation of models in both retrieval-based and generative tasks. Spanning six programming languages, it offers a robust benchmark for code-to-natural-language (Code-to-NL) generation. In our experiments, we primarily use the dataset for Java to train and evaluate our model.

3.5 Evaluation Metrics

For RQ1 and RQ2, we follow the methodology of previous studies (Shi et al. 2022, 2023; Wang et al. 2024; Yang et al. 2024), using accuracy to evaluate the model's performance on Vulnerability Prediction and Clone Detection tasks. For the Code Documentation Generation task, we assess the model's capabilities using BLEU and PPL scores. Regarding RQ3, our focus is on optimizing specific evaluation metrics. Thus, we select the PPL score as the optimization target for the Code Documentation Generation task. A brief description of each metric is provided below:

- **Accuracy:** A standard metric for classification tasks, accuracy measures the proportion of correctly predicted samples among all samples. It is used to evaluate the model's effectiveness in identifying vulnerabilities and detecting code clones.
- **BLEU:** The Bilingual Evaluation Understudy (BLEU) score is a metric for evaluating the quality of text generated by a model. It measures the overlap between n-grams in the generated output and reference texts, making it a popular choice for tasks involving natural language generation, such as code documentation.
- **PPL:** Perplexity (PPL) is a measure of how well a probability model predicts a sample. In the context of code documentation generation, a lower PPL score indicates that the model generates more coherent and contextually appropriate descriptions for the given code snippets.

3.6 Implementation Details

Our experiments are built on the PyTorch framework. We use the Hugging Face implementation of the studied PCMs. All training and evaluations are conducted on a single Ubuntu 20.04 server equipped with an Intel Xeon Gold 6226R CPU, 256GB RAM, and four NVIDIA GeForce RTX 3090 GPUs.

4 Results and Analysis

4.1 RQ1: Effects of Different Distillation Paradigms

4.1.1 Motivation

Knowledge distillation is a widely adopted and empirically effective model compression technique, valued for its simplicity and compatibility with diverse neural architectures. Over time, two primary KD paradigms have been developed: response-based and feature-

based. The former aligns model outputs, while the latter distills knowledge from intermediate representations such as hidden states and attention maps. While both paradigms have shown effectiveness in NLP tasks, their relative performance on pre-trained code models (PCMs) remains underexplored. This raises an important research question: *How do different KD paradigms affect the performance of PCMs across downstream tasks?*

4.1.2 Methodology

We first clarify the differences between two prominent knowledge distillation paradigm.

① **Response-based Distillation Paradigm.** Response-based distillation transfers knowledge by aligning the output logits of the student with those of the teacher. In our experiments, we implement this paradigm by computing the soft cross-entropy loss between the student's logits and the teacher's logits:

$$\mathcal{L}_{soft} = CE((z^T)/t, (z^S)/t) \tag{1}$$

where z^S and z^T are the logits vectors predicted by the student and teacher, respectively, CE represents the cross-entropy loss, and t denotes the temperature value. In our experiments, we set $t = 1$, in accordance with the recommendations of prior research (Shi et al. 2023, 2022; Jiao et al. 2020). In complex tasks, it is important to note that the knowledge provided by the teacher model, represented as soft labels, is not sufficient on its own. The ground truth labels of the data, referred to as hard labels, are also required as a supplementary component. Consequently, the response-based distillation loss function is defined as follows:

$$\mathcal{L}_{response} = \alpha CE((z^T)/t, (z^S)/t) + (1-\alpha)CE(\hat{y}, z^S) \tag{2}$$

where $\hat{y}$ represents the true label of the data, and α is introduced to adjust the weights between the hard labels and the soft labels.

② **Feature-based Distillation Paradigm.** Feature-based knowledge distillation transfers intermediate representations from the teacher model, such as hidden states, embedding layers, or feature maps, to the student model, enabling richer supervision beyond final output distributions. Among existing methods, TinyBERT (Jiao et al. 2020) has become one of the most widely adopted approaches. TinyBERT employs a two-stage distillation framework: a pre-training stage where knowledge is distilled during large-scale pre-training, followed by a task-specific stage where distillation is performed on downstream datasets. This design has shown remarkable effectiveness, with the six-layer $\mathrm{TinyBERT}_6$ achieving performance comparable to its teacher $\mathrm{BERT}_{\mathrm{BASE}}$, and the four-layer $\mathrm{TinyBERT}_4$ retaining over 96.8% of $\mathrm{BERT}_{\mathrm{BASE}}$ performance on the GLUE benchmark. In our experiments, however, we observed that pre-training stage distillation contributes little to downstream performance, a finding consistent with the ablation results reported in the original TinyBERT study. Given its limited impact and additional computational cost, we restrict our investigation to the task-specific distillation stage. In TinyBERT, task-specific distillation primarily comprises three components: Transformer-layer distillation, Embedding-layer distillation, and Prediction-layer distillation. We elaborate on each component in the following sections.

❶ ***Transformer-layer Distillation***. The proposed Transformer-layer distillation comprises attention-based and hidden-states-based methods. Attention-based distillation lever-

ages the pre-trained model's ability to encode rich linguistic knowledge, such as syntactic structures and semantic relationships. This approach facilitates knowledge transfer from the teacher to the student by aligning multi-head attention matrices, with the objective formulated as:

$$\mathcal{L}_{attn} = \frac{1}{h}\sum_{i=1}^{h} MSE(A_i^S, A_i^T) \tag{3}$$

where, *h* denotes the number of attention heads, A_i represents the attention matrix for the *i*-th head (teacher or student), and *MSE* is the mean squared error loss. In addition to attention-based distillation, this approach also distills knowledge from the hidden states of the Transformer layers, with the objective defined as:

$$\mathcal{L}_{hidn} = MSE(H^S W_h, H^T) \tag{4}$$

where the matrices H^S and H^T refer to the hidden states of student and teacher networks respectively, The matrix W_h is a trainable linear transformation that maps the student network's hidden states to the same space as the teacher network's states.

❷ ***Embedding-layer Distillation***. Similar to hidden-states distillation,the embedding-layer distillation, with the objective defined as:

$$\mathcal{L}_{embd} = MSE(E^S W_e, E^T) \tag{5}$$

where the matrices E^S and E^T refer to the embeddings of student and teacher networks respectively, The matrix W_e is a trainable linear transformation playing a similar role as W_h.

❸ ***Prediction-layer Distillation***. For the prediction layer, the loss calculation follows the same formula as in the response-based distillation paradigm, with the objective defined as:

$$\mathcal{L}_{pred} = \mathcal{L}_{response} \tag{6}$$

Using the above distillation objectives (i.e. Equations (3), (4), (5), (6)), The feature-based distillation loss between the teacher and the student network is defined as the summation of its individual components (embedding, hidden/attention, and prediction), each assigned an equal weight:

$$\mathcal{L}_{feature} = \begin{cases} \mathcal{L}_{embd} \\ \mathcal{L}_{hidn} + \mathcal{L}_{attn} \\ \mathcal{L}_{pred} \end{cases} \tag{7}$$

4.1.3 Experimental Design

For the RQ1 experiments, following the setup described in Section 3.3 and Section 3.4, we conduct experiments using CodeBERT, CodeT5+, and Qwen2.5-Coder across three differ-

ent code-related downstream tasks. Notably, in this experiments, we focus on two representative student model scales: **4-layer (small)** and **6-layer (medium)** models.

In the Transformer-layer Distillation stage of the feature-based distillation paradigm, the discrepancy between the number of layers in the student model and the teacher model necessitates the definition of a layer mapping function. Specifically, assuming the student model has *M* Transformer layers and the teacher model has *N* Transformer layers, the Transformer-layer distillation begins by selecting *M* layers from the *N* layers in the teacher model. A mapping function $n = g(m)$ is then defined to establish a correspondence between the indices of the student and teacher layers. In particular, this means that the *m*-th layer of the student model learns information from the *g*(*m*)-th layer of the teacher model. We set the layer mapping function in the experiment according to the TinyBERT paper. we use $g(m) = (M/N) \times m$. In short, for a 12-layer teacher model and a 6-layer student model, the student learns from the teacher by aligning with every two layers, thereby uniformly distilling the teacher's hidden representations. The specific layer mapping strategy is further explored in RQ2.

4.1.4 Results

Table 2 reports the performance of student models trained under different distillation paradigms across a range of downstream tasks. Overall, models distilled using either paradigm outperform those trained directly on the original dataset, validating the effectiveness of knowledge distillation for compressing PCMs. Notably, the *feature-based distillation* para-

Table 2 The experimental results of student models with two different sizes under various distillation paradigms. "*layer_response" represents models trained using the response-based distillation paradigm, "*layer_feature" denotes models trained using the feature-based distillation paradigm, and "*layer" indicates models trained directly on the original dataset. Best results are **underlined and bold**

Model	Vul(↑)	Clone(↑)	Doc-BLEU(↑)	Doc-PPL(↓)
CodeBERT				
Teacher	64.28	97.95	19.13	20.83
6layer_feature	**62.63**	**97.70**	**18.41**	**26.70**
6layer_response	61.86	96.45	17.09	37.17
6layer	59.66	96.25	17.03	37.95
4layer_feature	61.57	**96.75**	**16.45**	**44.22**
4layer_response	**61.60**	96.12	15.51	56.24
4layer	59.59	95.18	15.40	59.31
CodeT5+				
Teacher	64.93	97.32	9.02	37.99
6layer_feature	64.57	**96.65**	**5.18**	**184.10**
6layer_response	**64.75**	96.28	4.92	189.98
6layer	63.03	95.13	4.97	203.13
4layer_feature	**63.80**	**94.60**	**4.94**	**230.46**
4layer_response	63.29	93.85	4.85	235.54
4layer	62.26	93.45	4.73	250.92
Qwen2.5-Coder				
Teacher	63.95	96.70	18.96	11.81
6layer_feature	**62.96**	**96.07**	14.29	72.19
6layer_response	62.30	95.75	**16.85**	**30.79**
6layer	60.47	95.12	14.01	93.73
4layer_feature	**62.01**	**95.72**	13.89	75.07
4layer_response	61.02	94.88	**15.86**	**44.34**
4layer	59.81	93.08	13.90	97.59

digm exhibits consistent advantages over *response-based distillation*, particularly in semantically demanding tasks.

From the perspective of different model architectures, For both CodeBERT and CodeT5+, feature-based distillation consistently outperforms response-based distillation across all layer configurations. The most significant gains are observed in the code documentation generation task, where feature-based distillation yields an average 25% reduction in perplexity (PPL) and an 8% increase in BLEU score compared to response-based distillation, highlighting its superior capacity for capturing deep semantic representations via intermediate-layer alignment. However, an exception arises with Qwen2.5-Coder: in the documentation generation task, response-based distillation surprisingly surpasses feature-based distillation (e.g., a BLEU score of 16.85 compared to 14.29 in the 6-layer setting). This deviation suggests that model architecture may influence the effectiveness of distillation strategies, meriting further investigation.

From the perspective of different distillation approaches, we analyze the performance differences between feature-based and response-based knowledge distillation paradigms. While the feature-based paradigm consistently outperforms response-based distillation in vulnerability detection and code clone detection, the magnitude of improvement is comparatively limited, particularly for CodeT5+, where gains average less than 1% on these classification tasks. However, in code documentation generation tasks, feature-based knowledge distillation demonstrates more significant advantages, with improvement magnitudes reaching up to 28% on the PPL metric for the 6-layer student model. We attribute this to two factors. First, code documentation generation requires a deeper semantic understanding of source code. Feature-based distillation transfers richer intermediate representations from the teacher to the student, thereby enhancing semantic modeling, a capability especially critical for generative tasks. Second, vulnerability detection and code clone detection are binary classification tasks of relatively low complexity. Student models trained directly on labeled data can already achieve strong performance, leaving limited headroom for further improvement through distillation.

In summary, while feature-based distillation generally facilitates more effective knowledge transfer (Table 2), its performance is task-dependent and sensitive to model architecture. These findings underscore that feature-based distillation is not universally superior across all scenarios.

> ***Answer to RQ1***: All distillation paradigms improve student model performance across various tasks and architectures, despite their differences. **Feature-based distillation** generally outperforms **response-based distillation** by leveraging richer hidden-layer information. However, feature-based distillation is not universally superior in all settings.

4.2 RQ2 :The Impact of Different Parameter Configurations

4.2.1 Motivation

Through our investigation of different knowledge distillation paradigms, We find that feature-based distillation, although theoretically advantageous by leveraging intermediate-layer knowledge from the teacher model, does not consistently deliver superior perfor-

mance, as reflected in the RQ1 results. This inconsistency underscores the need for a deeper examination of the paradigm.

Feature-based distillation typically combines multiple processes (e.g., hidden state matching and loss weighting), each involving several hyperparameters that must be configured based on prior assumptions. However, the extent to which these configurations affect the overall effectiveness of knowledge distillation in PCMs remains unclear. This leads to the following research question: *What is the impact of different parameter configurations on the effectiveness of knowledge distillation in PCMs?*

4.2.2 Methodology

To answer RQ2, we focus on two parameters that may significantly influence the effectiveness of feature-based distillation: the choice of loss functions and the design of mapping functions. In the following, we discuss each parameter in detail and analyze how variations in their design affect the distillation process.

Exploring Different Attention Loss Functions The attention loss, denoted as $\mathcal{L}_{attn}$, measures the discrepancy between the attention distributions of the teacher and student models. The original paper (Jiao et al. 2020) method employs Mean Squared Error (MSE) on the unnormalized attention matrices. We investigate an alternative loss function, the Kullback-Leibler (KL) divergence, as illustrated in the following equation.

$$\frac{1}{h}\sum_{i=1}^{h} MSE(A_i^S, A_i^T) \rightarrow \frac{1}{h}\sum_{i=1}^{h} KL(\sigma(A^S)_i, \sigma(A_i^T)) \tag{8}$$

where, h denotes the number of attention heads, A_i represents the attention matrix for the i-th head (teacher or student), and σ represents softmax. We explore KL divergence because it measures the difference between two probability distributions, making it particularly suitable for comparing attention distributions.

Exploring Different Mapping Functions As described previously, the mapping functions determine which teacher layers a student layer learns from. In the original TinyBERT paper, a uniform mapping function $g(m) = (M/N) \times m$ is used. For instance, if the student model consists of 6 layers and the teacher model has 12 layers, the mapping coefficient is computed as $12/6 = 2$. In other words, the mapping is defined such that student layer 1 corresponds to teacher layer 2, student layer 2 to teacher layer 4, and so on. However, uniform mapping is not necessarily the optimal strategy. To this end, we design and evaluate three distinct mapping strategies—**Uniform**, **Bottom**, and **Top**—as illustrated in Fig. 3.

1. **Uniform:** The uniform mapping function is defined as $g(m) = K \times m$, where K denotes the ratio between the number of layers (N) in the teacher model and that in the student model (M). This formulation ensures that the student layers are evenly aligned with the teacher layers. For example, when distilling from a 12-layer teacher to a 6-layer student, the first student layer is mapped to the second teacher layer, the second student layer to the fourth teacher layer, and so forth, as illustrated in Fig. 3(a).

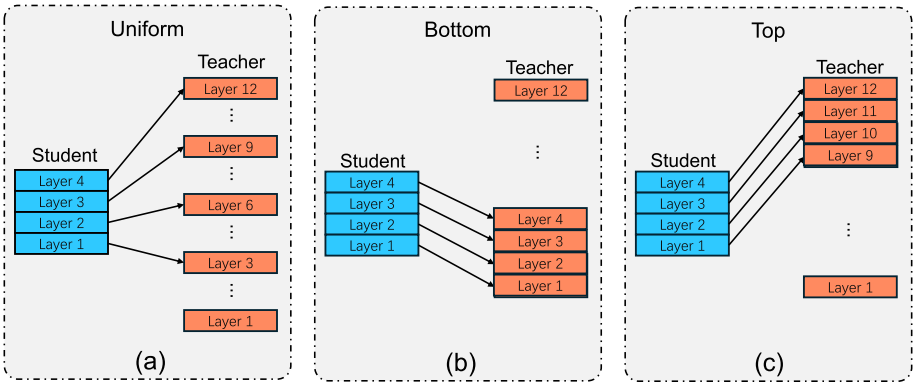

Fig. 3 the diagram of different layer mapping functions

2. **Bottom:** The bottom mapping function is defined as $g(m) = m$. This strategy focuses the student model's learning on the teacher layers closest to the word embedding layer. For instance, when distilling from a 12-layer teacher to a 6-layer student, the first student layer is aligned with the first teacher layer, the second student layer with the second teacher layer, and so forth, as illustrated in Fig. 3(b).
3. **Top:** The top mapping function is defined as $g(m) = m + M - N$. This strategy aligns the student's learning with the teacher layers closer to the logits output. For example, when distilling from a 12-layer teacher to a 6-layer student, the first student layer is mapped to the 7th teacher layer, the second student layer to the 8th teacher layer, and so forth, as illustrated in Fig. 3(c).

Exploring Different Numbers of Intermediate-layer Distillation Iterations Feature-based knowledge distillation typically consists of two stages: intermediate-layer distillation and prediction-layer distillation. For prediction-layer distillation, the training process can be monitored and terminated based on evaluation metrics once satisfactory performance is achieved. However, intermediate-layer distillation poses a unique challenge: since the output layer does not directly participate in this stage, evaluation metrics cannot reliably assess distillation quality. Consequently, the number of training epochs is typically determined empirically. For example, TinyBERT (Jiao et al. 2020) adjusts intermediate-layer distillation epochs based on downstream task complexity, with more difficult tasks requiring longer training. Nevertheless, this parameter remains difficult to specify precisely.

To investigate its impact on student model performance and training efficiency, we conduct a systematic experimental study. Specifically, we vary the number of intermediate-layer distillation epochs while keeping all other hyperparameters fixed. Our objective is to empirically identify the optimal epoch count that maximizes student model performance, thereby providing more reliable guidance for future knowledge distillation research.

4.2.3 Experimental Design

In this RQ, we conduct three controlled experiments to investigate the impact of key parameter configurations in feature-based distillation. For **attention loss formulations**, we compare Mean Squared Error (MSE) and Kullback–Leibler (KL) divergence across the three downstream tasks introduced earlier, with the layer mapping strategy fixed to Uniform. For **layer mapping strategies**, we evaluate three alternatives: Uniform, Bottom, and Top, with the attention loss function fixed to MSE, also across all three downstream tasks. For **intermediate-layer distillation iterations**, we focus specifically on the vulnerability detection task and systematically vary the number of training epochs for the intermediate-layer distillation stage across all three mapping strategies to assess the generalizability of our findings. In all experiments, we use the 6-layer student model described in Section 3.3 and ensure that loss functions and all other hyperparameters remain constant except for the variable being investigated, thereby enabling fair comparison.

4.2.4 Results

Table 3 reports the results of incorporating KL divergence as an alternative attention loss. Overall, the findings indicate that the effect of attention loss functions including Mean Squared Error (MSE) and Kullback–Leibler (KL) divergence varies considerably across downstream tasks and model architectures, with model structure playing a critical role in shaping their effectiveness. Notably, the CodeT5+ model demonstrates strong robustness to the choice of loss function, showing consistently marginal differences (all below 1%) across evaluation metrics. This suggests that, for CodeT5+, the optimization landscapes induced by MSE and KL divergence are similarly well-behaved. In contrast, the Qwen2.5-Coder model exhibits marked sensitivity to the choice of loss function. This effect is most pronounced in the code documentation generation task, where KL divergence substantially reduces perplexity (53.15 vs. 72.19 with MSE), yielding an absolute improvement of 19.04 and a relative gain of 26.4%.

From the perspective of task dependency, the relative advantages of different loss functions vary across tasks. For vulnerability detection, model performance is particularly sensitive to the choice of loss function, with differences in relative accuracy reaching up to 3%, a gap that, as shown in our RQ1 experiments, is non-negligible under the current experimental settings. For clone detection, the impact of loss function choice is comparatively minor. In code documentation generation, the influence of loss function extends beyond overall performance, shaping different evaluation metrics in divergent ways. Most notably,

Table 3 Performance comparison of different Loss Functions across three PCMs. Best results are **underlined and bold**

PCM	Loss Function	Vul(↑)	Clone(↑)	Doc-BLEU(↑)	Doc-PPL(↓)
Code-BERT	MSE	62.63	**97.70**	**18.41**	26.70
	KL	**63.54**	97.10	17.67	**25.90**
CodeT5+	MSE	**64.57**	**96.65**	5.18	**184.10**
	KL	63.47	96.62	**5.46**	184.74
Qwen2.5-Coder	MSE	62.96	96.07	14.29	72.19
	KL	**63.87**	**96.55**	**15.45**	**53.15**

KL divergence leads to substantially lower perplexity, especially on Qwen2.5-Coder. These observations underscore the multifaceted nature of sequence generation quality and highlight the necessity of aligning loss function selection with task-specific objectives and evaluation metrics. These findings empirically substantiate the theoretical insights discussed earlier. Specifically, the choice of attention loss function represents a critical hyperparameter in knowledge distillation, yet its effectiveness is inherently non-universal. Its utility is strongly conditioned on both the characteristics of the downstream task and the inductive biases embedded in the model architecture. Therefore, adopting a uniform strategy for loss function selection is suboptimal. Instead, performance can be maximized only through informed and context-sensitive tuning that aligns with the demands of the specific task and model configuration.

Table 4 reports the experimental results of evaluating different mapping functions. While the uniform strategy generally achieves the best overall performance, the Bottom strategy demonstrates competitive effectiveness under specific task settings. For example, on Qwen2.5-Coder it yields a 43% reduction in perplexity (PPL) compared to the uniform strategy. It is also worth emphasizing that the space of mapping functions is not limited to the three variants examined here. As the gap in layer counts between the student and teacher models widens, the number of feasible mapping configurations grows exponentially. These observations indicate that the choice of mapping function plays a critical role in determining the effectiveness of transformer distillation. Nevertheless, given the diversity of possible mappings, no single strategy can be expected to consistently outperform others across all tasks.

Figure 4 reports the systematic experimental results evaluating the impact of the number of intermediate-layer distillation iterations on student model accuracy. Across all investigated mapping strategies (Bottom, Uniform, Top), the student model's performance consistently shows a rapid initial increase followed by stabilization or a marginal decline. For example, the Bottom strategy achieves the peak accuracy of approximately 63.6% at 6 distillation rounds. It is crucial to note the consequences of under-specification: terminating the process prematurely at only 2 rounds results in a significant performance deficit (up to 2% lower than the peak), underscoring that insufficient training leads to substantially incomplete intermediate-layer knowledge transfer. Conversely, extending the training beyond the optimal point (e.g., to 14 rounds) yields negligible gains, demonstrating clear diminishing returns. These findings robustly confirm that the number of intermediate-layer distillation

Table 4 Performance comparison of different Mapping Functions across three PCMs. Best results are **underlined and bold**

PCM	Mapping Function	Vul(↑)	Clone(↑)	Doc-BLEU(↑)	Doc-PPL(↓)
CodeBERT	Uniform	62.63	**97.70**	18.41	26.70
	Bottom	**62.96**	97.10	**18.84**	**25.69**
	Top	62.12	96.02	17.94	29.86
CodeT5+	Uniform	**64.57**	**96.65**	5.18	**184.10**
	Bottom	62.59	96.25	**5.65**	184.85
	Top	62.41	96.08	4.96	185.09
Qwen2.5-Coder	Uniform	**62.96**	**96.07**	14.29	72.19
	Bottom	62.41	95.98	**17.40**	**29.77**
	Top	62.85	95.42	13.56	57.07

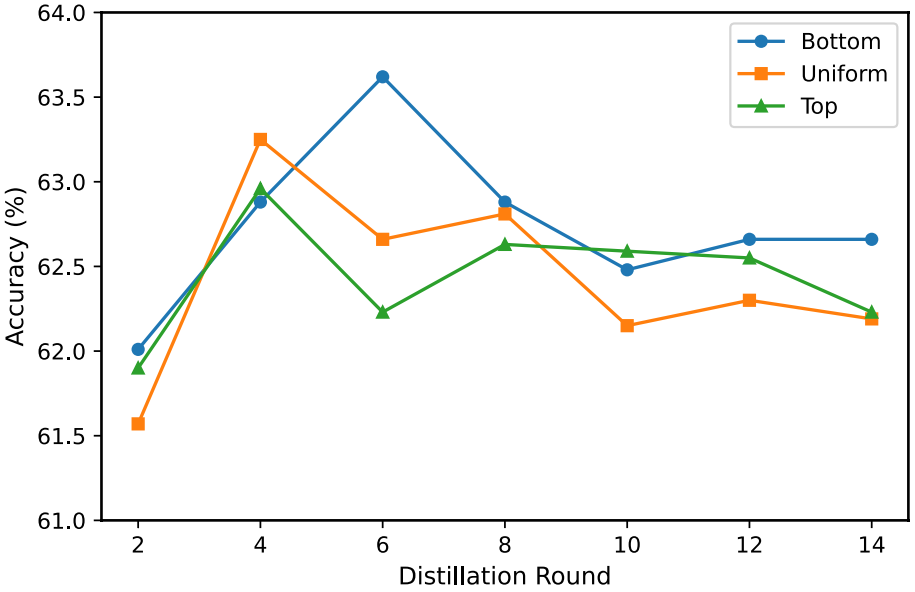

Fig. 4 Performance comparison on the vulnerability detection task with different numbers of intermediate-layer distillation iterations

iterations is a critical hyper-parameter whose precise specification plays a decisive role in determining the final effectiveness of the distillation experiment.

> ***Answer to RQ2***: The choice of **attention loss function** (MSE vs KL) significantly affects distillation performance, with task and model-specific sensitivities. The **number of intermediate-layer distillation iterations** is a critical hyper-parameter; systematic tuning is necessary as under-specification leads to substantial performance deficits, while over-specification provides diminishing returns. The **mapping function** configuration is critical; while Uniform performs well overall, Bottom can outperform it in specific cases (e.g., Qwen2.5-Coder). No single configuration is optimal across all settings, highlighting the need for task- and architecture-aware tuning.

4.3 RQ3: Enhance Knowledge Distillation Technique

4.3.1 Motivation

In Section 4.2, we observe that three critical configurations—the choice of the attention loss function, the layer mapping function, and the number of intermediate-layer distillation epochs—significantly impact the effectiveness of feature-based distillation paradigms. Although some studies have suggested that certain parameter choices exhibit advantages, these selections are often empirically determined and rarely demonstrate consistent performance across diverse code-related downstream tasks. In the context of compressing

PCMs, the ambiguity across these multiple critical settings poses substantial challenges for developers seeking predictable performance gains. Consequently, a critical research question arises: *Can the effectiveness of existing knowledge distillation techniques be enhanced through efficient parameter optimization?*

4.3.2 Methodology

Building on the findings in Section 4.2, we observe two critical distillation parameters that must be determined during the feature-based knowledge distillation phase: the choice of the loss function for computing Attention Loss and the configuration of the mapping function. For the loss function, the choice is formulated as a binary decision between Kullback–Leibler (KL) divergence and Mean Squared Error (MSE). For the mapping function, we seek an appropriate design to achieve optimal distillation performance. Specifically, we define a mapping function $g(m)$ that determines which teacher layers each student layer learns from. It is important to note that two prerequisites are imposed. First, not every layer of the student model is required to undergo distillation; each Transformer layer can either participate in the mapping process or be excluded. Second, cross-layer mappings are disallowed. Given that Transformer models propagate information sequentially layer by layer, we prohibit conflicting or reversed learning scenarios. Formally, for two student layers m_i and m_j where $m_i < m_j$, we require $0 < g(m_i) < g(m_j) < N$, where N denotes the total number of teacher layers.

Additionally, we make an important observation during the experimental process regarding the parameter configuration for intermediate-layer distillation. As demonstrated in our previous analysis, the number of intermediate-layer distillation epochs significantly impacts distillation effectiveness. However, since evaluation metrics cannot reliably assess intermediate-layer distillation quality during training, this parameter must be determined empirically. Therefore, we treat both the number of intermediate-layer distillation epochs and the learning rate as critical hyperparameters requiring systematic optimization. The learning rate is explored in conjunction with the epoch count, as these two parameters interact closely: a higher learning rate may enable faster convergence with fewer epochs, while a lower learning rate may require more epochs to achieve adequate feature alignment. By jointly optimizing these parameters within well-defined ranges, we aim to identify configurations that maximize both distillation effectiveness and training efficiency.

After identifying the parameters that influence the distillation process, to systematically determine the optimal distillation configuration for a given task and student model, we propose BOKD (*Bayesianly Optimized Intermediate-layer Knowledge Distillation*). While BOKD integrates established techniques, its technical innovation is two-fold and uniquely tailored for distilling PCMs: 1) It defines a specialized, high-dimensional hyperparameter search space that encompasses critical, sensitive parameters for code models (e.g., non-linear feature mapping strategies, precise intermediate-layer iteration depths) that are typically fixed or ignored in standard KD. 2) It introduces and leverages a novel, predictive "Effectiveness Indicator"—a cost-efficient surrogate model—that is built to quantify the

distillation potential of PCMs. This indicator is crucial for making the subsequent Bayesian optimization computationally feasible and highly effective in identifying the optimal distillation configuration across the entire configuration space.

Algorithm 1 Effectiveness Indicator.

Input: C: distillation hyperparameter search space; M: pretrained code model (teacher); D: training dataset; V: validation dataset; k: number of sampled configurations
Output: Effectiveness indicator model
1 $c \leftarrow \text{Sample}(C, k)$;
2 $e \leftarrow \varnothing$;
3 **for** $i \leftarrow 1$ **to** k **do**
4 $\quad N_i \leftarrow \text{INITIALIZE}(c_i)$;
5 $\quad N_i \leftarrow \text{FEATUREDISTILL}(M, N_i, D)$;
6 $\quad e_i \leftarrow \text{EVALUATE}(N_i, V)$;
7 **return** $\text{BAYES}(\{(c_i, e_i)\}_{i=1}^{k})$

We first need to clarify that the knowledge distillation process is a time-consuming procedure. Therefore, a key challenge is the prohibitive computational cost of evaluating each configuration through full training and testing. To mitigate this, BOKD introduces an auxiliary module, the **Effectiveness Indicator**, which acts as a lightweight surrogate model to approximate distillation performance without exhaustive computation. Concretely, as shown in Algorithm 1, we first sample k configurations from the search space C (line 1) and initialize an empty effectiveness set e (line 2). Then, for each sampled configuration c_i within the loop (line 3), the student model N_i is initialized according to c_i (line 4), trained via feature-based distillation from the teacher model M on the training dataset D (line 5), and evaluated on the validation dataset V to obtain an empirical effectiveness score e_i (line 6). Finally, the configuration–performance pairs $(c_i, e_i)_{i=1}^{k}$ are used to fit a Bayesian Ridge Regression (BRR) model (Tew et al. 2023), which predicts expected distillation effectiveness across the entire configuration space (line 7). This approach allows efficient estimation of configuration quality without the need for exhaustive computation, substantially reducing the overall optimization cost (Huang et al. 2025).

For the optimization engine, BOKD employs the Hyperopt library in Python, which uses the Tree-structured Parzen Estimator (TPE)—a state-of-the-art Bayesian optimization algorithm tailored for black-box functions (Liao et al. 2022). TPE efficiently balances exploration and exploitation in the configuration space, enabling BOKD to converge toward high-performing settings with minimal overhead. The complete BOKD Pipeline including the construction of the Effectiveness Indicator, the parameter optimization process, and the final distillation is illustrated in Algorithm 2. Specifically, BOKD first constructs the Effectiveness Indicator $\mathcal{I}$ using Algorithm 1 to provide a lightweight surrogate model for evaluating candidate configurations (line 1). The history set $\mathcal{P}$ is initialized as empty to record explored configurations and their predicted effectiveness (line 2), and an initial configuration h_0 is randomly sampled from the search space C (line 3). Then, for each Bayesian optimization iteration i from 1 to N (line 4), BOKD predicts the expected effectiveness s_i of the current configuration h_{i-1} using the Effectiveness Indicator $\mathcal{I}$ (line 5). The pair (h_{i-1}, s_i) is added to the history set $\mathcal{P}$ (line 6), and a Bayesian surrogate model is fitted to $\mathcal{P}$ to guide the next configuration suggestion (line 7). Based on this surrogate, a new configuration h_i is suggested by the TPE algorithm (line 8). After completing N iterations, BOKD selects the configuration c^* with the highest predicted effectiveness from $\mathcal{P}$ (line 9) and returns it as the

optimal distillation hyperparameter setting (line 10). This design enables efficient and principled exploration of the hyperparameter space while leveraging the Effectiveness Indicator to reduce the computational cost of full training evaluations.

Algorithm 2 BOKD: Bayesianly Optimized Intermediate-layer Knowledge Distillation.

Input: M: pretrained code model (teacher); C: hyperparameter search space; k: number of samples for indicator construction; N: maximum BO iterations
Output: c^*: optimal distillation hyperparameter configuration

1 $\mathcal{I} \leftarrow$ BUILDEFFECTIVENESSINDICATOR(C, M, k) ; // Algorithm 1
2 $\mathcal{P} \leftarrow \emptyset$; // Initialize history set
3 $h_0 \leftarrow$ RANDOMSAMPLE(C) ; // Initial configuration
4 **for** $i \leftarrow 1$ **to** N **do**
5 $s_i \leftarrow \mathcal{I}$.PREDICT$(h_{i-1})$; // Evaluate via indicator
6 $\mathcal{P} \leftarrow \mathcal{P} \cup \{(h_{i-1}, s_i)\}$;
7 surrogate $\leftarrow$ FITBAYESSURROGATE$(\mathcal{P})$;
8 $h_i \leftarrow$ BO_SUGGEST$(surrogate, C)$;
9 $c^* \leftarrow \arg\max_{(h,s)\in\mathcal{P}} s$;
10 **return** c^* ;

4.3.3 Experimental Design

In our experiments, we apply the BOKD optimization process to three downstream tasks, using two primary evaluation metrics: accuracy and perplexity (PPL). To ensure the robustness and generalizability of the optimization results, we consider three student model sizes—specifically, the 2-layer, 4-layer, and 6-layer architectures described in Section 3.3. For the number of sampled configurations K, we set it to 20 based on a trade-off between optimization efficiency and computational cost. The number of Bayesian optimization iterations was fixed at 500 to ensure sufficient exploration of the configuration space.

4.3.4 Results

The experimental results, as summarized in Table 5, demonstrate the effectiveness of the proposed BOKD framework in enhancing the performance of distilled student models across various downstream tasks. After applying Bayesian optimization to determine the optimal distillation configurations, the student models exhibited substantial performance improvements compared to those distilled using conventional empirical settings. From the perspective of downstream tasks, the most substantial improvements are observed in the code documentation generation task, particularly in the perplexity (PPL) metric. On average, significant improvements are achieved across different student model sizes after applying BOKD. The 6-layer student model shows a 27% reduction in PPL, the 4-layer student model achieves a 19% reduction, and the 2-layer student model demonstrates a 29% reduction, with an average reduction of 25% across all student model sizes. For the vulnerability detection task, the performance gains are also notable, with the 2-layer student model distilled from CodeBERT achieving a 3% increase in accuracy, representing a 5% relative improvement compared to methods without parameter optimization. In contrast, for the clone detection task, only marginal improvements are observed across all models. This may

Table 5 Performance comparison of different distillation methods across various PCMs and model sizes. Best results per setting are highlighted with **bold underline**. Δ shows BOKD's improvement over Feature KD baseline

PCM	Size	KD Method	PPL (↓)	Vul (↑)	Clone (↑)	Δ vs Feature KD		
						PPL	Vul	Clone
CodeBERT		Feature KD	26.70	62.63	97.70	-	-	-
	6L	Response KD	37.17	61.86	96.45	-39.2%	-1.2%	-1.3%
		BOKD	**23.53**	**63.43**	**97.95**	+11.9%	+1.3%	+0.3%
		Feature KD	44.22	61.57	96.75	-	-	-
	4L	Response KD	56.24	61.60	96.12	-27.2%	+0.0%	-0.7%
		BOKD	**39.48**	**62.99**	**97.37**	+10.7%	+2.3%	+0.6%
		Feature KD	97.54	60.54	94.27	-	-	-
	2L	Response KD	99.22	59.88	94.70	-1.7%	-1.1%	+0.5%
		BOKD	**63.94**	**63.46**	**95.10**	+34.4%	+4.8%	+0.9%
CodeT5+		Feature KD	184.10	64.57	96.65	-	-	-
	6L	Response KD	189.98	**64.75**	93.88	-3.2%	+0.3%	-2.9%
		BOKD	**173.62**	64.31	**96.95**	+5.7%	-0.4%	+0.3%
		Feature KD	230.46	63.80	94.60	-	-	-
	4L	Response KD	235.54	63.29	95.18	-2.2%	-0.8%	+0.6%
		BOKD	**221.87**	**64.46**	**95.22**	+3.7%	+1.0%	+0.7%
		Feature KD	322.24	58.20	94.42	-	-	-
	2L	Response KD	316.42	58.86	94.10	+1.8%	+1.1%	-0.3%
		BOKD	**303.99**	**60.47**	**95.17**	+5.7%	+3.9%	+0.8%
Qwen2.5		Feature KD	72.19	62.96	96.07	-	-	-
-Coder	6L	Response KD	30.79	62.30	95.75	+57.3%	-1.0%	-0.3%
		BOKD	**26.08**	**63.64**	**97.20**	+63.9%	+1.1%	+1.2%
		Feature KD	75.07	62.01	95.72	-	-	-
	4L	Response KD	44.34	61.02	94.88	+40.9%	-1.6%	-0.9%
		BOKD	**42.80**	**63.49**	**96.15**	+43.0%	+2.4%	+0.4%
		Feature KD	146.77	61.46	94.67	-	-	-
	2L	Response KD	138.89	59.33	94.20	+5.4%	-3.5%	-0.5%
		BOKD	**76.86**	**63.06**	**96.08**	+47.6%	+2.6%	+1.5%

be due to the already strong baseline performance of knowledge distillation on this task, leaving limited room for further optimization benefits.

In terms of model architecture, BOKD demonstrates greater effectiveness when applied to encoder-only architectures such as CodeBERT and decoder-only architectures like Qwen2.5-Coder. On the code generation task with Qwen2.5-Coder, perplexity decreases by an average of 51% after parameter optimization. In comparison, the performance gains on the encoder-decoder architecture CodeT5+ are relatively modest. One possible explanation is that the architectural complexity and multi-objective training nature of encoder-decoder models may make the distillation process less sensitive to individual parameter tuning, thereby limiting the impact of configuration-level optimizations.

Overall, these results validate the design of BOKD, demonstrating that automated and task-aware optimization of key distillation parameters such as loss functions, feature mapping strategies, and iteration counts can significantly enhance the effectiveness of knowledge distillation across diverse tasks and student model capacities.

> ***Answer to RQ3***: The results show that BOKD effectively enhances knowledge distillation by automatically optimizing key parameters. It delivers substantial gains on tasks such as code documentation generation (up to 50% reduction in perplexity) and notable improvements in vulnerability detection, while maintaining competitive performance on clone detection.

5 Discussions

5.1 Efficiency of Our Approach

The proposed parameter optimization framework comprises two main phases. In the first phase, a set of candidate distillation configurations is sampled, and their effectiveness is evaluated by training student models and measuring downstream task performance using metrics such as accuracy or PPL, depending on the task. This step is the primary contributor to the overall time cost.

For instance, in our experiments on the CodeBERT model, constructing the Effectiveness Indicator for a 6-layer student model with 20 sampled configurations using a single 24GB NVIDIA GeForce RTX 3090 typically requires approximately 11 hours. In contrast, for a 2-layer student model, the same process usually takes less than 4 hours. Crucially, this overhead is incurred only once per unique teacher-student architecture pair. Once the Indicator is constructed, it can be reused (amortized) for subsequent optimizations across all relevant downstream tasks, making the long-term cost negligible for deployment-ready models.

In the second phase, Bayesian optimization is employed to identify the optimal parameter setting. Since the Effectiveness Indicator has already been built in the first phase, this step is computationally lightweight and incurs negligible time overhead. Even with 500 optimization iterations, BOKD typically converges to an optimal configuration within approximately 2 minutes, demonstrating both efficiency and practical scalability.

The initial sample size, k, is a key design choice representing a trade-off between the indicator's accuracy and the initial computational cost. Our choice of $k = 20$ was determined to strike a practical balance ensuring sufficient exploration of the complex high-dimensional search space while keeping the setup cost feasible (e.g., the 11-hour cost for the 6-layer CodeBERT student).However, the number of sampling iterations can be flexibly configured to accommodate available computational resources and time constraints. To further enhance efficiency, practitioners can choose to reduce k. Through our validation, in the code documentation generation task, reducing the initial sampling quantity from $k = 20$ to $k = 10$ does not severely affect the correctness of the effectiveness indicator while saving 50% of the time, proving the feasibility of adjusting k based on resource availability. Additionally, the training process may be performed on a carefully selected subset of the full dataset, which preserves the relative ranking of configuration effectiveness with minimal compromise in accuracy. This data-efficient strategy facilitates faster convergence during optimization while ensuring reliable and robust performance estimates, rendering the proposed method both practical and scalable for real-world deployment scenarios.

To provide practical guidance for deploying compressed models in real-world applications, we evaluate the inference efficiency of our student models obtained through BOKD.

Table 6 Inference time comparison between Teacher Models and Compressed Student Models. Latency is measured in milliseconds (ms). The percentage in parentheses indicates the compression speed-up rate

Model (Size)	Latency (ms)		
	Vulnerability Prediction	Clone Detection	Code Doc Generation
CodeBERT (126M)	1012	1875	2649
2-Layer Student (7.8M)	241(**76.19 %**)	382(**79.63%**)	592 (**77.65 %**)
CodeT5+ (220 M)	1764	2731	3533
2-Layer Student (7.9 M)	341 (**80.67 %**)	462 (**83.08 %**)	674 (**80.92 %**)
Qwen2.5-Coder (494M)	2676	4651	5743
2-Layer Student (16.3M)	584 (**78.18 %**)	649 (**86.05 %**)	1091 (**81.00 %**)
Average Improvements	**78.34 %**	**82.92 %**	**79.86 %**

Following the deployment scenario where models are integrated into IDEs or code editors as plugins, Following the experimental setup described in Section 3.6, we measure the inference latency by limiting the model to use only 8 CPU cores to simulate typical laptop environments. We set the input lengths consistent with RQ1 experiments: 400 tokens for vulnerability prediction, 800 for clone detection, and 1024 for code documentation generation. For each task, we randomly sample 100 examples from the test set and repeat the experiments three times to reduce randomness effects.

Table 6 presents the inference latency comparison between teacher models and our 2-layer compressed student models. The results demonstrate substantial efficiency improvements across all three tasks and teacher models. For CodeBERT compression, our 2-layer student model (7.8M parameters) achieves an average speedup of 76.19%, 79.63%, and 77.65% on the three tasks respectively, reducing latency from over 1000ms to around 200-600ms. Similar improvements are observed for CodeT5+ and Qwen2.5-Coder, with average speedup rates of 80.67-83.08% and 78.18-86.05% respectively. Notably, on the clone detection task with longer inputs (800 tokens), our compressed models achieve the highest average speedup of 82.92%, demonstrating their efficiency advantage on processing longer sequences. These results confirm that BOKD-compressed models not only maintain competitive performance but also provide significant inference efficiency benefits, making them highly suitable for deployment in resource-constrained environments such as developer laptops and IDEs.

5.2 BOKD in the Model Compression Landscape

While our work focuses on optimizing Knowledge Distillation (KD), it is crucial to contextualize BOKD's results within the broader model compression landscape, particularly against pruning and quantization. KD excels at achieving high performance retention even at aggressive compression ratios (e.g., reducing a 12-layer teacher to a 2-layer student), as demonstrated by our models retaining 96-97% of teacher performance. This is because KD restructures the entire network and is not limited by the original architecture. In contrast, pruning removes redundant weights or entire heads, which can shrink the model footprint but often fails to reduce the size of the embedding layer—a significant component in many PCMs—and can require specialized hardware or complex sparse matrix multiplication

libraries for real-world inference speedup. Quantization, which reduces precision (e.g., from FP32 to INT8), offers excellent inference speedup and immediate memory reduction but is highly dependent on specific hardware (CPU/GPU) support and can lead to immediate and significant accuracy drops if not carefully applied, particularly to PCMs where numerical precision in structured logic may be crucial. BOKD, through its automated optimization of the KD pipeline, provides a pathway to high-quality, dense student models that are easy to deploy and offer substantial speedup and memory reduction compared to the teacher, making it a highly effective and complementary compression strategy.

5.3 Threats to Validity

Regarding internal validity, our experiments primarily focused on compressing pre-trained models through different knowledge distillation strategies and optimized distillation hyperparameters. However, the performance of the compressed student models may also be influenced by other residual hyperparameters, such as input sequence length or the number of training epochs. To mitigate this threat, we kept all non-distillation hyperparameters consistent with settings reported in prior work (Shi et al. 2022) when fine-tuning the teacher models. We further verified that our re-trained teacher models achieved performance comparable to or better than those reported in the literature (Shi et al. 2023), ensuring that the compression was based on well-trained teachers. During the distillation process, we only varied the distillation-specific parameters under investigation, while maintaining all other settings consistent within each architecture. Moreover, a potential risk lies in the correctness of the distillation pipeline implementations. To minimize this, we carefully validated and reviewed all experiment scripts.

In terms of external validity, we selected three representative downstream tasks: vulnerability detection, clone detection, and code documentation generation. While this set does not cover all possible code-related tasks, it spans classification, similarity detection, and generation paradigms, providing a broad evaluation spectrum. Additionally, we evaluated our approach across three distinct model architectures: the encoder-only CodeBERT, the decoder-only Qwen2.5-Coder, and the encoder-decoder CodeT5+. This diversity helps strengthen the generalizability of our findings.We explicitly acknowledge three additional threats:

1. *Dataset Bias:* The evaluation datasets, while standard benchmarks, may not fully capture the complexity and diversity of real-world, industry-specific codebases (e.g., legacy systems, obscure languages, highly specialized APIs). This potential dataset bias may limit the generalizability of our absolute performance gains to every real-world deployment scenario.
2. *Architecture-Specific Findings:* Although we tested three distinct model families, the parameter sensitivities we identify may not directly transfer to entirely new, radically different architectures (e.g., future state-space models or novel decoder-only families). Our findings are most robust for Transformer-based architectures common in PCMs.
3. *Task-Specific Optimization Scope:* The BOKD Effectiveness Indicator is trained and validated on the downstream tasks presented. Its reliable predictive capacity may degrade if applied to a deployment task that is significantly different in nature, domain, or data

distribution from the tasks used to construct the indicator. Future work is needed to explore the Indicator's cross-task transferability.

Despite these threats, the consistent improvements observed across the evaluated tasks and architectures suggest that our proposed optimization method is broadly applicable to a wide range of large code models and code-related downstream applications.

6 Related Work

In recent years, both the natural language processing and software engineering communities have focused on optimizing large language models to balance performance and efficiency. Existing model compression studies can be broadly divided into three main categories: model pruning, model quantization, and knowledge distillation.

Model pruning can be divided into unstructured and structured pruning (Ganesh et al. 2021; Cheng et al. 2024). The former works by replacing some model parameters with zero, In contrast, structured pruning removes larger components such as layers or attention heads, providing hardware-efficient acceleration. Gordon et al. (2020) show that pruning 30–40% of BERT's parameters does not harm downstream accuracy. FLAP (An et al. 2024) introduces a retraining-free structured pruning framework for large language models, achieving hardware-friendly compression and faster inference while outperforming prior pruning methods. CoFi (Xia et al. 2022) introduces task-specific structured pruning that jointly prunes layers, heads, and hidden units, achieving over $10\times$ speedups with accuracy comparable to distillation. These results highlight the complementarity between structured pruning and knowledge distillation, motivating their integration in our future work.

Model quantization reduces model size by representing parameters in lower-bit formats (e.g., 8-bit or binary) instead of 32-bit floats, enabling efficient inference on resource-constrained devices with minimal performance loss (Kuzmin et al. 2023). ZeroQuant (Yao et al. 2022) achieves up to $5\times$ inference speedup with minimal accuracy loss through post-training quantization for Transformer models. SmoothQuant (Xiao et al. 2023) enables 8-bit quantization with up to $1.56\times$ speedup and $2\times$ memory reduction while preserving accuracy. However, models cannot be quantized to very small bit ranges, and quantization methods cannot achieve significant acceleration effects for model inference.

We have discussed knowledge distillation techniques for pre-trained models in Section 2.2 and introduced TinyBERT (Jiao et al. 2020). In software engineering and code intelligence, distillation has also been applied to compress PCMs. For example, Compressor (Shi et al. 2022) successfully reduced a 500 MB model to just 3 MB, though it did not explore improvements to the distillation process itself. Similarly, Avatar (Shi et al. 2023) focused on the environmental impact and carbon footprint of student models, but paid limited attention to refining distillation mechanisms. To our knowledge, our work is the first to systematically investigate and optimize various knowledge distillation strategies for large PCMs.

7 Conclusion and Future Work

This paper presents a comprehensive study on the effectiveness of knowledge distillation for compressing large PCMs. Our investigation spans three representative downstream tasks, including code vulnerability detection, code clone detection, and code documentation generation, and involves three distinct model architectures: the encoder-only CodeBERT, the decoder-only Qwen2.5-Coder, and the encoder-decoder CodeT5+. Experimental results indicate that feature-based knowledge distillation generally achieves better performance than response-based approaches across multiple tasks and model architectures; however, there remains considerable room for improvement, particularly in optimizing its effectiveness for certain tasks and model types.

Furthermore, we identify several key factors that critically influence the performance of feature-based distillation, including the choice of loss function for attention alignment, the mapping function used in intermediate-layer distillation, and the number of epochs dedicated to hidden state distillation. Building on these insights, we propose an automated parameter optimization framework, BOKD, which leverages Bayesian optimization to search for optimal distillation configurations. Our approach achieves significant performance gains, improving student model performance by up to 50% on the code documentation generation task.

In addition, regarding scalability to foundation models, the optimization logic of BOKD is inherently model-agnostic and operates at the level of distillation hyperparameters rather than model-specific architectures. Therefore, it is theoretically scalable to ultra-large models (e.g., beyond 7B or 70B parameters), as well as to emerging multi-modal code models, provided that the corresponding distillation signals (e.g., hidden states or cross-modal representations) can be properly defined. Future work will focus on validating this scalability in large-scale settings and extending the framework to more complex model paradigms.

Author Contributions Weifeng Sun and Ruifeng Wu contributed equally to this work as co-first authors. These two authors performed the study conception, design, material preparation, and data analysis. The manuscript was jointly written and revised by them. All authors participated in the discussion, read, and approved the final manuscript.

Funding This work was supported in part by the National NaturalScience Foundation of China under Grant No. 62372071, in part by the Fundamental Research Funds for the Central Universities under GrantNo. 2022CDJDX-005, in part by the Chongqing Technology Innovation and Application Development Project under Grant Nos. CSTB2022TIADSTX0007and CSTB2023TIAD-STX0025, in part by the Natural Science Foundation of Sichuan Province under Grant No. 2026NSFSC1454, and inpart by the "Large-Scale Microservice Resource Dynamic Optimization Technology Cooperation" project of Mashang Consumer Finance Co., Ltd.

Data Availability Statement The datasets generated during and/or analyzed during the current study are available in the [KDcode2public] repository, https://github.com/Waylandite/KDcode2public.

Declarations

Conflict of Interest The authors declared that they have no conflict of interest.

Ethical Approval This declaration is not applicable for this study.

Informed Consent This declaration is not applicable for this study.

Clinical Trial Number This declaration is not applicable for this study.

References

Al Debeyan F, Hall T, Madeyski L (2025) Emerging results in using explainable AI to improve software vulnerability prediction. In: Proceedings of the 33rd ACM international conference on the foundations of software engineering, pp 561–565

An Y, Zhao X, Yu T, Tang M, Wang J (2024). Fluctuationbased adaptive structured pruning for large language models. In: Proceedings of the AAAI conference on artificial intelligence. Vol 38. 10, pp 10865–10873

Ba J, Caruana R (2014) Do deep nets really need to be deep? Adv Neural Inf Process Syst 27

Cheng H, Zhang M, Shi JQ (2024) A survey on deep neural network pruning: taxonomy, comparison, analysis, and recommendations. IEEE Transactions on Pattern Analysis and Machine Intelligence

Cho JH, Hariharan B (2019) On the efficacy of knowledge distillation. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 4794–4802

Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805

Dohmke T, Iansiti M, Richards GL (2023) Sea change in software development: economic and productivity analysis of the AI-powered developer lifecycle. https://api.semanticscholar.org/CorpusID:259261950

Dvivedi SS, Vijay V, Rahul Pujari SL, Lodh S, Kumar D (2024). A comparative analysis of large language models for code documentation generation. In: Proceedings of the 1st ACM international conference on AI-powered software, pp 65–73

Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, Shou L, Qin B, Liu T, Jiang D, Zhou M (2020) CodeBERT: a pre-trained model for programming and natural languages. In: Cohn T, He Y, Liu Y (eds) Findings of the association for computational linguistics: EMNLP 2020. Online: Association for Computational Linguistics, pp 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

Feng S, Suo W, Wu Y, Zou D, Liu Y, Jin H (2024) Machine learning is all you need: a simple token-based approach for effective code clone detection. In: Proceedings of the IEEE/ACM 46th international conference on software engineering, pp 1–13

Ganesh P, Chen Y, Lou X, Khan MA, Yang Y, Sajjad H, Nakov P, Chen D, Winslett M (2021) Compressing large-scale transformer-based models: a case study on bert. Trans Assoc Comput Linguist 9:1061–1080

GitHub Copilot Community (2023) GitHab. GitHub Copilot Community. https://github.com/orgs/community/

Gong Y, Liu L, Yang M, Bourdev L (2014) Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115

Gordon MA, Duh K, Andrews N (2020). Compressing bert: studying the effects of weight pruning on transfer learning. arXiv preprint arXiv:2002.08307

Gou J, Baosheng Yu, Maybank SJ, Tao D (2021) Knowledge distillation: a survey. Int J Comput Vision 129(6):1789–1819

Guo D, Ren S, Lu S, Feng Z, Tang D, Liu S, Zhou L, Duan N, Svyatkovskiy A, Fu S et al (2020). Graphcodebert: pretraining code representations with data flow. arXiv preprint arXiv:2009.08366

Han S, Mao H, Dally WJ (2015a) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149

Han S, Pool J, Tran J, Dally W (2015b) Learning both weights and connections for efficient neural network. Adv Neural Inf Process Syst 28

Hinton G (2015) Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531

Huang Y, Li Y, Wu W, Zhang J, Lyu MR (2023) Do not give away my secrets: Uncovering the privacy issue of neural code completion tools. arXiv preprint arXiv:2309.07639

Huang L, Sun W, Yan M (2025) Iterative Generation of Adversarial Example for Deep Code Models. In: 2025 IEEE/ACM 47th international conference on software engineering (ICSE). IEEE Computer Society, pp 623–623

Hui B, Yang J, Cui Z, Yang J, Liu D, Zhang L, Liu T, Zhang J, Yu B, Dang K et al (2024) Qwen2. 5-Coder Technical Report. arXiv preprint arXiv:2409.12186

Husain H, Wu H-H, Gazit T, Allamanis M, Brockschmidt M (2019) Codesearchnet challenge: evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436

Jiao X, Yin Y, Shang L, Jiang X, Chen X, Li L, Wang F, Liu Q (2020) TinyBERT: distilling BERT for natural language understanding. In: Cohn T, He Y, Liu Y (eds) Findings of the association for computational linguistics: EMNLP 2020. Online: Association for Computational Linguistics, pp.4163–4174. https://doi.org/10.18653/v1/2020.findingsemnlp.372

Kanade A, Maniatis P, Balakrishnan G, Shi K (2020). Learning and evaluating contextual embedding of source code. In: International conference on machine learning. PMLR, pp 5110–5121

Karmakar A, Robbes R (2021) What do pre-trained code models know about code? In: 2021 36th IEEE/ACM international conference on automated software engineering (ASE), pp 1332–1336. https://doi.org/10.1109/ASE51524.2021.9678927

Kuzmin A, Nagel M, Van Baalen M, Behboodi A, Blankevoort T (2023) Pruning vs quantization: which is better? Adv Neural Inf Process Syst 36:62414–62427

Liao L, Li H, Shang W, Ma L (2022) An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. ACM Trans Softw Eng Methodol (TOSEM) 31(3):1–40

Lin B, Wang S, Liu Z, Liu Y, Xia X, Mao X (2023) CCT5: a code-change-oriented pre-trained model. 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: Association for Computing Machinery, pp 1509– 1521. https://doi.org/10.1145/3611643.3616339. isbn: 9798400703270

Liu Y, Sheng L, Shao J, Yan J, Xiang S, Pan C (2018) Multi-label image classification via knowledge distillation from weakly-supervised detection. In: CoRR

Lo D (2023) Trustworthy and synergistic artificial intelligence for software engineering: Vision and roadmaps. In: 2023 IEEE/ACM international conference on software engineering: future of software engineering (ICSE-FoSE). IEEE, pp 69–85

Lu S, Guo D, Ren S, Huang J, Svyatkovskiy A, Blanco A, Clement CB, Drain D, Jiang D, Tang D, Li G, Zhou L, Shou L, Zhou L, Tufano M, Gong M, Zhou M, Duan N, Sundaresan N, Deng SK, Fu S, Liu S (2021) CodeXGLUE: a machine learning benchmark dataset for code understanding and generation. arXiv:2102.04664

Luo Q, Ye Y, Liang S, Zhang Z, Qin Y, Lu Y, Wu Y, Cong X, Lin Y, Zhang Y et al. (2024). Repoagent: an llm-powered open-source framework for repository-level code documentation generation. arXiv preprint arXiv:2402.16667

Mastropaolo A, Scalabrino S, Cooper N, Palacio DN, Poshyvanyk D, Oliveto R, Bavota G (2021) Studying the usage of text-to-text transfer transformer to support code-related tasks. In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE). IEEE, pp 336–347

Mirzadeh SI, Farajtabar M, Li A, Levine N, Matsukawa A, Ghasemzadeh H (2020) Improved knowledge distillation via teacher assistant. In: Proceedings of the AAAI conference on artificial intelligence. Vol 34. 04, pp 5191–5198

Nijkamp E, Pang B, Hayashi H, Tu L, Wang H, Zhou Y, Savarese S, Xiong C (2022). Codegen: an open large language model for code with multi-turn program synthesis. arXiv preprint arXiv:2203.13474

Niu C, Li C, Luo B, Ng V (2022) Deep Learning Meets Software Engineering: A Survey on Pre-Trained Models of Source Code. In: De Raedt L (ed) Proceedings of the thirty-first international joint conference on artificial intelligence, IJCAI-22. Survey Track. International Joint Conferences on Artificial Intelligence Organization, pp 5546–5555. https://doi.org/10.24963/ijcai.2022/775

Niu C, Li C, Ng V, Chen D, Ge J, Luo B (2023a) An empirical comparison of pre-trained models of source code. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp 2136–2148. https://doi.org/10.1109/ICSE48619.2023.00180.

Niu L, Mirza S, Maradni Z, Pöpper C (2023b) CodexLeaks: privacy leaks from code generation language models in GitHub copilot. In: 32nd USENIX Security Symposium (USENIX Security 23), pp 2133–2150

Phuong M, Lampert C (2019) Towards understanding knowledge distillation. In: International conference on machine learning. PMLR, pp 5142– 5151

Romero A, Ballas N, Kahou SE, Chassang A, Gatta C, Bengio Y (2014) Fitnets: hints for thin deep nets. arXiv preprint arXiv:1412.6550

Sanh V, Debut L, Chaumond J, Wolf T (2019) DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108

Shi J, Yang Z, Xu B, Kang HJ, Lo D (2022) Compressing pre-trained models of code into 3 mb. In: Proceedings of the 37th IEEE/ACM international conference on automated software engineering, pp 1–12

Shi J, Yang Z, Kang HJ, Xu B, He J, Lo D (2023). Smaller, faster, greener: compressing pre-trained code models via surrogateassisted optimization. arXiv preprint arXiv:2309.04076

Sun Z, Yu H, Song X, Liu R, Yang Y, Zhou D (2020) MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. In: Jurafsky D, Chai J, Schluter N, Tetreault J (eds) Proceedings of the 58th annual meeting of the association for computational linguistics. Online: Association for Computational Linguistics, pp 2158–2170. https://doi.org/10.18653/v1/2020.acl-main.195

Svajlenko J, Islam JF, Keivanloo I, Roy CK, MiaMM (2014) Towards a big data curated benchmark of inter-project code clones. In: 2014 IEEE international conference on software maintenance and evolution. IEEE, pp 476–480

Svyatkovskiy A, Lee S, Hadjitofi A, Riechert M, Franco JV, Allamanis M (2021) Fast and memory-efficient neural code completion. In: 2021 IEEE/ACM 18th international conference on mining software repositories (MSR). IEEE, pp 329–340

Tang R, Lu Y, Liu L, Mou L, Vechtomova O, Lin J (2019) Distilling task-specific knowledge from bert into simple neural networks. arXiv preprint arXiv:1903.12136

Tew SY, Boley M, Schmidt D (2023) Bayes beats cross validation: efficient and accurate ridge regression via expectation maximization. Adv Neural Inf Process Syst 36:19749–19768

Wang Y, Wang W, Joty S, Hoi SCH (2021) Codet5: identifieraware unified pre-trained encoder-decoder models for code understanding and generation. arXiv preprint arXiv:2109.00859

Wang Y, Le H, Gotmare AD, Bui NDQ, Li J, Hoi SCH (2023) Codet5+: open code large language models for code understanding and generation. arXiv preprint arXiv:2305.07922

Wang H, Tang Z, Tan SH, Wang J, Liu Y, Fang H, Xia C, Wang Z (2024) Combining structured static code information and dynamic symbolic traces for software vulnerability prediction. In: Proceedings of the IEEE/ACM 46th international conference on software engineering, pp 1–13

Wei X, Gonugondla SK, Wang S, Ahmad W, Ray B, Qian H, Li X, Kumar V, Wang Z, Tian Y et al (2023) Towards greener yet powerful code generation via quantization: an empirical study. In: Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering, pp 224– 236

Xia M, Zhong Z, Chen D (2022) Structured pruning learns compact and accurate models. arXiv preprint arXiv:2204.00408

Xiao G, Lin J, Seznec M, Wu H, Demouth J, Han S (2023).SmoothQuant: accurate and efficient post-training quantization for large language models. In: Krause A, Brunskill E, Cho K, Engelhardt B, Sabato S, Scarlett J (eds) Proceedings of the 40th international conference on machine learning. Vol 202. Proceedings of Machine Learning Research. PMLR, pp 38087–38099. https://proceedings.mlr.press/v202/xiao23c.html

Xu C, Zhou W, Ge T, Wei F, Zhou M (2020) BERT-ofTheseus: compressing BERT by progressive module replacing. In: Webber B, Cohn T, He Y, Liu Y (eds) Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP). Online: Association for Computational Linguistics, pp 7859–7869. https://doi.org/10.18653/v1/2020.emnlp-main.633

Xu FF, Alon U, Neubig G, Hellendoorn VJ (2022) A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN international symposium on machine programming, pp 1–10

Xu X, Li M, Tao C, Shen T, Cheng R, Li J, Xu C, Tao D, Zhou T (2024) A survey on knowledge distillation of large language models. arXiv preprint arXiv:2402.13116

Yang Z, Zhao Z, Wang C, Shi J, Kim D, Han D, Lo D (2023) What do code models memorize? An empirical study on large language models of code 10. arXiv preprint arXiv:2308.09932

Yang Y, Xing H, Gao Z, Chen J, Ni C, Xia X, Lo D (2024) Federated learning for software engineering: A case study of code clone detection and defect prediction. IEEE Trans Softw Eng 50(2):296–321

Yang A, Li A, Yang B, Zhang B, Hui B, Zheng B, Yu B, Gao C, Huang C, Lv C, Zheng C, Liu D, Zhou F, Huang F, Hu F, Ge H, Wei H, Lin H, Tang J, Yang J, Tu J, Zhang J, Yang J, Yang J, Zhou J, Zhou J, Lin J, Dang K, Bao K, Yang K, Yu L, Deng L, Li M, Xue M, Li M, Zhang P, Wang P, Zhu Q, Men R, Gao R, Liu S, Luo S, Li T, Tang T, Yin W, Ren X, Wang X, Zhang X, Ren X, Fan Y, Su Y, Zhang Y, Zhang Y, Wan Y, Liu Y, Wang Z, Cui Z, Zhang Z, Zhou Z, Qiu Z (2025) Qwen3 Technical Report. arXiv preprint arXiv:2505.09388

Yao Z, Aminabadi RY, Zhang M, Xiaoxia W, Li C, He Y (2022) Zeroquant: efficient and affordable post-training quantization for large-scale transformers. Adv Neural Inf Process Syst 35:27168–27183

Yoon JW, Lee H, Kim HY, Cho WI, Kim NS (2021) TutorNet: towards flexible knowledge distillation for end-to-end speech recognition. IEEE/ACM Trans Audio Speech Lang Process 29:1626–1638

Zadeh AH, Edo I, Awad OM, Moshovos A (2020) GOBO: quantizing attention-based NLP models for low latency and energy efficient inference. In: 2020 53rd Annual IEEE/ACM international symposium on microarchitecture (MICRO), pp 811–824. https://doi.org/10.1109/MICRO50266.2020.00071

Zeng Z, Tan H, Zhang H, Li J, Zhang Y, Zhang L (2022) An extensive study on pre-trained models for program understanding and generation. In: Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis. ISSTA 2022. Virtual, South Korea: Association for Computing Machinery, pp 39–51. https://doi.org/10.1145/3533767.3534390. isbn: 9781450393799

Zhang W, Hou L, Yin Y, Shang L, Chen X, Jiang X, Liu Q (2020). TernaryBERT: distillation-aware Ultra-low Bit BERT. In: Webber B, Cohn T, He Y, Liu Y (eds) Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP). Online: Association for Computational Linguistics, pp 509–521. https://doi.org/10.18653/v1/2020.emnlp-main.37

Zhang Z, Chen C, Liu B, Liao C, Gong Z, Yu H, Li J, Wang R (2023) survey on language models for code. arXiv preprint arXiv:2311.07989

Zhou Y, Liu S, Siow J, Du X, Liu Y (2019) Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks. Adv Neural Inf Process Syst 32
Zhu X, Li J, Liu Y, Ma C, Wang W (2024) A survey on model compression for large language models. Trans Assoc Comput Linguist 12:1556–1577

Authors and Affiliations

Weifeng Sun[1] · Ruifeng Wu[1] · Hongyan Li[1] · Ying Fu[3] · Min Yu[2] · Meng Yan[1]

✉ Meng Yan
mengy@cqu.edu.cn

Weifeng Sun
weifeng.sun@cqu.edu.cn

Ruifeng Wu
wuruifeng@stu.cqu.edu.cn

Hongyan Li
hongyan.li@cqu.edu.cn

Ying Fu
YingFu@swjtu.edu.cn

Min Yu
27930946@qq.com

[1] School of Big Data and Software Engineering, Chongqing University, Chongqing, China

[2] Mashang Consumer Finance Co., Ltd., Chongqing, China

[3] School of Computing and Artificial Intelligenc, Southwest Jiaotong University, Chongqing, China