MANTRA: a Framework for Multi-stage Adaptive Noise TReAtment During Training

ZIXIAO ZHAO, University of British Columbia, Canada
FATEMEH H. FARD, University of British Columbia, Canada
JIE JW WU, Michigan Technological University, United States

The reliable application of deep learning models to software engineering tasks hinges on high-quality training data. Yet, large-scale repositories inevitably introduce noisy or mislabeled examples that degrade both accuracy and robustness. While Noise Label Learning (NLL) has been extensively studied in other fields, there are a few works that investigate NLL in Software Engineering (SE) and Large Language Models (LLMs) for SE tasks. In this work, we propose **MANTRA**, a **M**ulti-stage **A**daptive **N**oise **TR**e**A**tment framework that embeds noise diagnosis and mitigation directly into the fine-tuning process of code-Pretrained Language Models (PTM) and code-LLMs.

We first investigate the effect of noise at varying levels on convergence and loss trajectories of the models. Then we apply an adaptive dropout strategy guided by per-sample loss dynamics and Gaussian Mixture Model clustering to exclude persistently noisy points while preserving clean data. Applying to code summarization and commit intent classification, our experiments reveal that some LLMs are more sensitive to noise than others. However, with MANTRA, the performance of all models in both tasks is improved. MANTRA enables researchers and practitioners to reduce the impact of errors introduced by the dataset in training, saves time in data cleaning and processing, while maximizing the effect of fine-tuning.

CCS Concepts: • **Software and its engineering**; • **Computing methodologies** → **Natural language processing**;

Additional Key Words and Phrases: Noise label learning, LLM, software engineering, code summarization, commit intent classification

ACM Reference Format:
Zixiao Zhao, Fatemeh H. Fard, and Jie JW Wu. 2024. MANTRA: a Framework for Multi-stage Adaptive Noise TReAtment During Training. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (October 2024), 21 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The swift progress of Artificial Intelligence (AI) in Software Engineering (SE) has driven the adoption of deep learning models across multiple SE tasks [1, 4, 36, 39, 41]. These models, which include Code-Pretrained Language Models (PTM) and Code-Large Language Models (LLMs), are utilized for different software engineering tasks like code summarization and bug detection [33]. The effectiveness of these models greatly depends on the quality of the training data, since noisy data can cause incorrect predictions, poorer generalization, and unreliable results [6, 12, 38, 42, 49]. Studies have shown that smaller models can achieve close or even better performance than larger

Authors' addresses: Zixiao Zhao, zixiaosh@student.ubc.ca, University of British Columbia, Kelowna, BC, Canada; Fatemeh H. Fard, fatemeh.fard@ubc.ca, University of British Columbia, Kelowna, Canada; Jie JW Wu, jie.jw.wu@mtu.edu, Michigan Technological University, Houghton, Michigan, United States.

1049-331X/2024/10-ART1 $15.00
https://doi.org/XXXXXXX.XXXXXXX

models when trained on a higher-quality dataset [15]. However, obtaining high-quality, noise-free datasets for code-related tasks remains an ongoing challenge [29, 40, 46], especially as these datasets are frequently derived from large-scale platforms like GitHub and StackOverflow, where manual verification is infeasible due to data volume and complexity [29].

Given the inherent complexity of program data, noisy data exists across different benchmarks [42], such as when descriptions fail to match paired code snippets for code summarization [22] or a fully functional code is mislabeled as buggy [48]. The presence of noise in the training data can cause slow convergence and gradient instability [42], affecting models' robustness in real-world applications [14, 27]. Robustness in this context is defined as the ability of models to maintain performance when confronted with noisy data.

To tackle these challenges, many studies have concentrated on Noise Label Learning (NLL) to create methods for handling noise in datasets [24, 25, 54, 57], with the goal of improving model robustness by identifying and mitigating the impact of erroneous labels [29, 57]. However, NLL studies in SE are limited in number and scope. They are mostly empirical studies or applied on classification tasks [7, 9, 29, 42, 45], with no NLL approach in SE that works for a variety of tasks like classification and generative tasks. Additionally, these methods tend to be optimized for small to medium-sized models, leaving a gap in effective noise management techniques for LLMs in software engineering. On the other hand, the NLL research in other fields typically rely on techniques that require pre-calculated confidence estimates [26], or dedicated or task-specific models for noise handling [2, 17]. While some researchers have proposed self-training techniques to handle noisy data during training [29, 53], to the best of our knowledge, no work in SE has addressed self-training for different task types in various-sized models, including PTMs and LLMs.

Our research seeks to fill these gaps by proposing a generalized framework for managing noisy data in code-related tasks, focusing on PTMs and LLMs to improve model robustness in the presence of noisy data. In this work, we present **MANTRA**, a ***M**ulti-stage **A**daptive **N**oise **TR**e**A**tment* designed specifically to tackle the issues of noisy labels and data anomalies in tasks related to SE. This study comprises two core stages:

- **Investigation of Noise Effects on Model Dynamics:** We conduct a systematic investigation into how noise influences model convergence and loss trajectories during training, introducing synthetic noise at different levels to simulate real-world data inconsistencies. This contribution provides novel insights into how noisy data points affect model stability.
- **Dynamic Noise Management via Adaptive Dropout Strategies:** We develop a dynamic method for identifying and dropping noise-prone data points during fine-tuning. This approach combines adaptive dropout with noise-resistant regularization techniques, offering a targeted solution for mitigating noise effects without compromising data diversity. This contribution is particularly valuable for fine-tuning in real-world applications, where clean datasets are rare.

We experiment with five different code models, both from PTM and LLMs: CodeBERT, CodeT5+, CodeLlama-7B-HF, StarCoder2-7B, and Qwen2.5-Coder-7B. The models are assessed on two SE tasks: (i) *code summarization*, and (ii) *commit intent classification*. Each model is evaluated in four label noise settings for each task: 0%, 5%, 10%, and 15%, with and without MANTRA. Our findings support that PTMs have different loss distributions for clean and noisy data [29]; however, this finding depends on the studied LLMs. CodeLlama shows different distributions for clean and noise samples, but, this is less pronounced for the other two LLMs. Additionally, some LLMs are more robust to noisy data than others. In all models, both PTMs and LLMs, MANTRA makes the models robust, with minimal performance degradation in the presence of label noise.

MANTRA offers a task-agnostic way to spot and correct label noise as training unfolds, strengthening the reliability and robustness of large code models without per-task retuning. Restraining noise at each stage of learning yields more accurate and generalizable systems and sets the groundwork for future research on noise-resilient training in SE applications.

The primary contributions of this research are: First, by analyzing loss trajectories across two code intelligence tasks for PTMs and LLMs, we shed new light on how label noise behaves. Second, we introduce MANTRA, a model and task-agnostic framework that detects and down-weights suspected noisy instances *during* fine-tuning. Finally, our experiments demonstrate that MANTRA consistently strengthens all five models in code summarization and commit-intent classification. We also provide a replication package for our study: https://anonymous.4open.science/r/MANTRA-3653/.

2 RELATED WORKS

Studies in software engineering and AI fields have investigated issues concerning data quality [6, 12, 49] and their effects on model performance [3], and investigating approaches to ensure high quality data [35]. Sample-selection methods filter out noise by estimating noise rates and confidence thresholds [37], calculating network-based p-values framed as a multiple-hypothesis testing problem [52], applying unsupervised methods such as K-Nearest Neighbor to enhance the model's ability to detect noise [57], using methods like causal analysis [54] and co-training [28], or applying thresholds to separate noise and clean data [38].

Other approaches [24, 26] focus on estimating noise generation matrices to minimize the noise generation process, or provide solvable conditions for noisy data in regression [11]. Other researchers employed Bayesian methods to improve estimation accuracy [50] or proposed new loss functions to improve robustness during training [31].

Bai et al. [2], Liu et al. [30] strengthen noise robustness by refining loss functions and employing early stopping to prevent overfitting to noise. Co-teaching involves one model identifying noise patterns in the labels and giving feedback to the other model, helping it better distinguish between noisy and clean samples [17, 28]. Yuan et al. [53] explored collaborative frameworks where small models interact with LLMs to identify and relabel noisy samples dynamically during training. Ye et al. [51] explore refinement of LLM-generated labels during training to calibrate pre-trained classifiers against label noise. Kim et al. [25] has used two EM procedures, enabling optimization under label noise without requiring external supervision. Hyperspherical margin weighting to adaptively reweight samples based on their angular distances in feature space is proposed [55]. Luo et al. [34] use a multi-expert system, context-enhanced denoising, and entropy-based sample selection to detect and relabel noisy responses. Wang et al. [47] introduces a framework that injects noise into a poisoning expert module during training to improve the robustness.

The studies on noise label learning in SE, however, is limited compared to the NLP and computer vision fields. An empirical study on three SE tasks, program classification, vulnerability detection, and code summarization, is conducted, comparing NLL approaches on smaller and larger models on different noise types [45]. Another study uses the semantic difference between samples to detect noise for the security defect task [7]. Data influence methods, Influence Function, and TracIn, are applied to detect and remove noisy samples in the source code corpus for two classification tasks [9]. A two-stage robust training framework is proposed in [29] to solve the problem of mislabeling and class imbalance in code-based classification datasets. Finally, a recent empirical study investigated data bugs across different data modalities, revealing their impact on training dynamics, thereby offering actionable insights for data cleaning and model monitoring [42].

Differences of our work. The closest studies to our approach are [29, 38]. The former [38] tailors GMM for classification tasks in NLP and erases the original labels of the identified samples for semi-supervised self-training. The latter [29] applies GMM on similarity scores, which need to

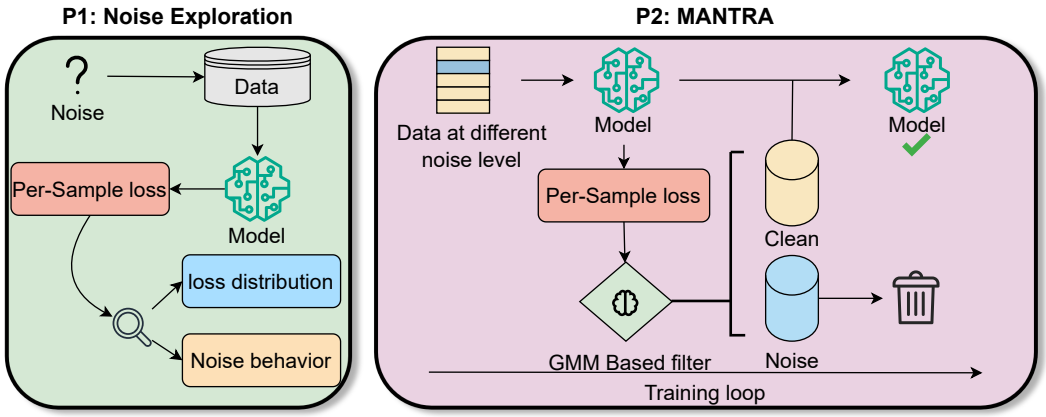


Fig. 1. The workflow for this study. P1 explores loss distribution and noise behaviour, P2 applies MANTRA during training.

be calculated based on ground truth or class representative for SE classification tasks. Furthermore, the earlier works [29, 38, 45] only use smaller deep learning and pre-training language models (e.g., Roberta, CodeBERT, UniXCoder), without exploring code large language models. In contrast, MANTRA works for classification *and* generative tasks, applied on PTMs and LLMs.

3 METHODOLOGY

In this section, we will examine the research questions, the tasks and models analyzed, and our method for addressing each research question.

3.1 Research Questions

We investigate the answer to the following questions.

RQ1: How does noise affect the loss trajectory and convergence behavior of PTMs and LLMs during training?

Understanding how noise influences training dynamics is essential for designing robust models capable of handling noisy data. Noisy samples can disrupt the learning process by causing inconsistent loss patterns, slower convergence, or even overfitting to erroneous labels [25]. Earlier studies indicate that clean and noisy samples exhibit different patterns in loss values and convergence rates [38]. This research question seeks to investigate the impact of noise on the tasks being studied, specifically for PTMs and LLMs. We introduce different levels of synthetic noise and track the loss trajectories of samples during training.

RQ2: How does MANTRA affect the performance of the models in the presence of noise?

This RQ investigates the effectiveness of MANTRA in improving robustness to noisy labels during fine-tuning. MANTRA identifies noise samples exhibiting inconsistent behavior and gradually excludes them via an adaptive dropout mechanism. This delay in exclusion ensures initial exposure to diverse data while minimizing long-term overfitting to corrupted labels. We compare the performance of the models trained both with and without MANTRA.

3.2 Approach

Figure 1 presents our approach. The first part (RQ1) explores the loss distribution of the models, and the second part (RQ2) applies MANTRA on the studied tasks. Details of each part are below.

3.2.1 RQ1: To address the first research question on how noise impacts loss trajectory and convergence behavior, we begin by introducing controlled noise into the datasets of each task randomly. Noise is added to the training portion of the datasets for each task, whereas the validation and test sets are left unchanged. We apply noise at three levels of intensity, 5%, 10%, and 15%, and consider 0% noise as the clean data. Here, 5%, 10%, and 15% denote the proportion of label corruption applied only to the training split, with class priors preserved; 0% is the clean control. This enables us to assess how varying levels of noise affect the model's performance. During training, we monitor loss trajectories across epochs for both clean and noisy samples. Observing fluctuations and inconsistencies in the loss values helps identify distinctive patterns associated with noisy data, such as slower or inconsistent loss reduction for noisy samples. Choosing 5%, 10%, and 15% label corruption lets us pinpoint the noise threshold at which each model's performance starts to break down and measure how well adaptive filtering methods recover accuracy as corruption increases. Additionally, we perform a convergence analysis by comparing the model's rate of convergence on datasets with varying noise densities, examining how different levels of noise affect the model's stability in reaching optimal performance. Losses are used to quantify the impact of noise. This analysis provides insights into how different ratios of noisy data influence the model's training dynamics, laying the groundwork for adaptive noise handling strategies in subsequent phases.

3.2.2 RQ2: The second research question explores whether dynamically identifying and dropping noisy samples during fine-tuning can enhance model robustness. We establish criteria to recognize noise-prone samples based on their loss trajectories. During fine-tuning, we apply an adaptive dropout strategy, removing samples that show persistent noise-like behavior after a specific number of epochs. This approach balances the model's exposure to diverse data with the need to avoid overfitting to noisy samples. By selectively removing samples only after the model has had an initial opportunity to learn from them, we enable the model to generalize more effectively. We then compare the performance of the model versus when it was trained without dynamic dropout.

To identify the noisy data, we use Gaussian Mixture Models (GMM) and Bayesian Information Criterion (BIC). GMMs are employed to determine whether the dataset comprises multiple underlying clean distributions. GMMs model the data as a mixture of Gaussian components, with each component defined by its own mean, variance, and weight. The parameters of these components are estimated using the Expectation-Maximization (EM) algorithm, which iteratively maximizes the likelihood of the data under the model.

E-step: $w_j^{(i)} = p\big(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma\big)$ and

M-step:

$$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)}, \qquad \mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} \big(x^{(i)} - \mu_j\big)\big(x^{(i)} - \mu_j\big)^{\top}}{\sum_{i=1}^{m} w_j^{(i)}}.$$

To select the optimal number of components in the GMM, we use the Bayesian Information Criterion, a measure that balances goodness-of-fit and model complexity. BIC is defined as: $\text{BIC} = -2\ln L + k\ln n$, where L is the likelihood of the model given the data, k represents the number of parameters, and n denotes the number of data points. The BIC applies a more substantial penalty for the number of parameters, which makes it especially effective for selecting simpler models when dealing with large datasets.

The BIC values are computed for GMMs with varying numbers of components, and the model with the lowest BIC is selected as the optimal representation of the data. This ensures a balance

between accurately capturing the underlying distributions and avoiding overfitting, providing a robust method to identify mixtures of normal distributions in the dataset.

3.3 Tasks and Datasets

We choose different tasks, including generative and classification, which represent the generalizability of MANTRA.

Code Summarization aims to generate summaries of source code [13, 16, 56], it has been studied in several papers [8, 10, 21, 43, 45], therefore chosen in our study. We use the Python subset of CodeSearchNet [21] dataset for our work. To make synthesized noise, replace the sample's code summary with randomly selected tokens, preserving the length of the summary unchanged. For fine-tuning LLMs on code summarization, we randomly selected 10,000 samples from the CodeSearchNet [21] Python dataset due to resource limitations. The validation and test sets remain the same.

Commit Intent Classification seeks to infer the purpose of a commit to a code change, for example, Bug, from its textual and code context. Accurate intent labels help reviewers prioritize urgent fixes, generate release notes, and support traceability throughout the CI/CD pipeline [18]. For our experiments, we use the dataset in [44], which contains commits whose titles, descriptions, changed-file paths, and diff hunks are manually annotated with one or more of seven intents: Bug, Refactor, Deprecation, Feature, Merge, Resource, and Test. To insert noise, we randomly replace the labels with a different intent. After filtering, we split the dataset into train, validate, and test using an 8:1:1 ratio, resulting in 700, 85, and 88 in the train, validate, and test datasets, respectively.

In both datasets, noisy data is replaced with the same ratio of clean data, thus, the dataset statistics remain the same.

3.4 Models

We selected five models of PTMs and LLMs selected in our study, informed by a previous work [36], to assess MANTRA's effectiveness. **CodeBERT** [13] is an encoder-only Transformer model that has demonstrated strong results across code-related tasks [1, 46] and is used in previous SE noise label learning studies [45]. **CodeT5+** [46] designed for code understanding and generation. **CodeT5+** [46] supports encoder-only, decoder-only, and encoder-decoder architectural, achieving state-of-the-art performance on more than 20 code benchmarks at the time of its release. **CodeLlama-7B-HF** [41] is a well-known series of large language models tailored specifically for code understanding and generation across multiple programming languages, built on the Transformer architecture [41]. **StarCoder2** [32] is a widely-used LLM for code, trained on The Stack v2, across multiple programming languages, and archived encouraging result. **Qwen2.5-Coder** [20] is the latest code-specialized branch of the Qwen2.5 family and has performance gains in code intelligence and is studied in various SE research [5, 23]. All five models are open source with checkpoints available from HuggingFace, and detailed specifications are provided in Table 1.

3.5 Evaluation Metrics

We explain the evaluation metrics used for each task in this section.

3.5.1 Code Summarization. We employ **BLEU-4** (Bilingual Evaluation Understudy for four-gram matches) as the primary metric to evaluate our code summarization approach. BLEU-4 quantifies how closely a generated summary aligns with a reference summary by comparing overlapping word sequences (*n*-grams) of lengths 1 to 4.

3.5.2 Code Commit intention classification. We employ the **micro averaged $\mathbf{F_1}$ score** (Micro-F1) to evaluate our multilabel commit intent classification. Micro F1 score aggregates the total true

Table 1. Pre-trained backbones and fine-tuning setup used in this study. All checkpoints were obtained from HuggingFace and are open-source.

Model	#Params	Context	FT method
CodeBERT	0.12 B	514	Full FT
CodeT5+	0.22 B	512	Full FT
CodeLlama-7B-HF	7.0 B	16 384	LoRA (r=16, $\alpha = 16$)
StarCoder2-7B	7.0 B	16 384	LoRA (r=16, $\alpha = 16$)
Qwen2.5-Coder-7B	7.61 B	32 768	LoRA (r=16, $\alpha = 16$)

positives (TP), false positives (FP), and false negatives (FN) over all classes and examples, then computes a single precision and recall before taking their harmonic mean:

For class i we define:

$\text{Precision}_i = TP_i/(TP_i + FP_i)$ and $\text{Recall}_i = TP_i/(TP_i + FN_i)$.

Aggregating counts over all n classes gives the micro-averaged metrics:

$\text{Precision}_{\text{micro}} = \left(\sum_{i=1}^{n} TP_i\right) / \left(\sum_{i=1}^{n} (TP_i + FP_i)\right)$ and $\text{Recall}_{\text{micro}} = \left(\sum_{i=1}^{n} TP_i\right) / \left(\sum_{i=1}^{n} (TP_i + FN_i)\right)$.

Their harmonic mean yields the micro-F_1:

$$F1_{\text{micro}} = 2 \cdot \frac{\text{Precision}_{\text{micro}} \cdot \text{Recall}_{\text{micro}}}{\text{Precision}_{\text{micro}} + \text{Recall}_{\text{micro}}}.$$

Because it weights every individual prediction equally, Micro-F1 provides a single, unified measure of overall classification quality, particularly well-suited to imbalanced or multi-label settings, such as commit intent analysis.

3.6 Experiment setup

All experiments were run on a GPU node with 1 CPU core and an A100 40 GB GPU. We used the hyper-parameter configurations specified in the original model papers, training the models with a learning rate of 0.00005.

In MANTRA, we choose to drop the samples after 3 epochs for code summarization and 5 epochs for commit intent classification. These numbers are set based on our observations from RQ1. We use the scikit-learn[1] library for all GMM-related parts.

4 RESULTS

In this section, we will present the findings of our research questions, followed by an analysis of the observations. We present the results of each of the two tasks separately in the following.

4.1 Loss Trajectory and Convergence Behavior in Presence of Noise

4.1.1 Code Summarization. Table 2 shows the average loss for code summarization for CodeBERT and CodeT5+, with various noise ratios over 10 epochs. The results reveal consistent patterns in the loss trajectories of these models during training, with variations based on the level of noise introduced into the dataset. In all noise levels, the loss decreases over epochs. However, the rate of this reduction and the final loss values are strongly influenced by the level of noise, showing the detrimental effects of noise on training.

The loss reduction for noisy data is slower compared to clean data. For instance, with 5% noise, the loss at epoch 1 for noisy data is 5.36 for CodeBERT, while for clean data it is 4.86. This gap

[1]https://scikit-learn.org/stable/index.htmlhttps://scikit-learn.org/stable/index.html

Noise	Type	1	2	3	4	5	6	7	8	9	10
CodeBERT											
5%	Noise	5.36	4.61	4.45	4.31	4.27	4.19	4.14	4.10	4.02	4.00
	clean	4.86	3.42	3.01	2.80	2.59	2.44	2.32	2.20	2.14	2.05
10%	Noise	5.39	4.59	4.33	4.24	4.17	4.10	4.02	4.00	3.93	3.90
	clean	4.85	3.62	3.22	3.09	2.86	2.69	2.53	2.45	2.34	2.29
15%	Noise	5.30	4.56	4.29	4.21	4.13	4.04	3.94	3.87	3.88	3.84
	clean	4.97	3.74	3.42	3.20	3.00	2.85	2.62	2.55	2.55	2.40
CodeT5+											
5%	Noise	1.74	0.57	0.55	0.54	0.55	0.52	0.55	0.51	0.55	0.54
	clean	2.14	0.30	0.27	0.25	0.23	0.21	0.19	0.18	0.15	0.14
10%	Noise	1.96	0.54	0.54	0.54	0.54	0.52	0.54	0.53	0.54	0.53
	clean	2.05	0.34	0.31	0.31	0.31	0.29	0.27	0.28	0.24	0.23
15%	Noise	1.82	0.55	0.54	0.53	0.54	0.52	0.53	0.54	0.53	0.54
	clean	2.08	0.36	0.35	0.33	0.32	0.31	0.33	0.33	0.30	0.29

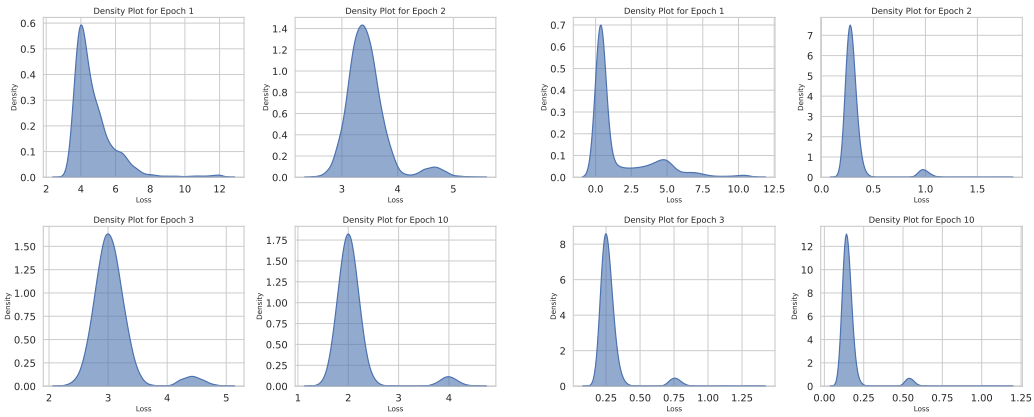
Table 2. Average loss for code summarization for CodeBERT and CodeT5+ at different noise levels (5%, 10%, 15%) randomly inserted into the dataset. Each row compares the loss for noisy data (Noise) and non-noisy data (clean) across epochs, shown with numbers 1–10. clean means 0% noise.

widens by epoch 10, where the losses are 4.00 and 2.05, respectively. Same trend is observed for 10% and 15% noise levels, and for CodeT5+ model (though in the first epoch, CodeT5+ has higher loss for clean data). This widening gap over epochs indicates that while the model can still converge on noisy data, the process becomes more challenging due to the presence of noise. The model requires additional epochs to effectively minimize loss, suggesting that even small amounts of noise can impede learning. Overall, the clean data has lower loss in the last epoch compared to the noisy data.

Finding 1: *For code summarization, the presence of noise leads to higher final losses and slower convergence for PTMS.*

Loss Density Plots for Code Summarization. The density plots in Figures 2a and 2b illustrate the evolution of the loss distribution across different epochs for CodeBERTand CodeT5+, highlighting the models' optimization process, over different epochs in presence of 5% noise. In the first epoch, the distribution is wide and relatively flat, indicating high variance in loss values. By epoch 2, the distribution becomes more peaked, reflecting reduced variance and a shift toward lower loss values as the model starts learning meaningful patterns. This trend continues in later epochs, where the density curve narrows further, and the peak moves closer to lower loss values. By epoch 10, the loss distribution is sharply peaked, with most values concentrated near the lower end of the scale, indicating that the model has effectively minimized loss for the majority of samples. This progressive narrowing and shifting of the density plots demonstrates the model's ability to reduce variance and improve consistency across samples during training. Any remaining width in the density plot or secondary peaks at later epochs could suggest the influence of noisy samples or outliers, which the model may struggle to optimize fully.

Finding 2: *For both PTMs, we observe two distinct distributions after the second epoch in their loss density plots for code summarization, which we relate to losses for non-noisy and noisy data.*



(a) Distribution of loss for CodeBERT (5%, epochs 1, 2, 3, 10)

(b) Distribution of loss for CodeT5+ (5%, epochs 1, 2, 3, 10)

Fig. 2. Distribution of loss for CodeBERT and CodeT5+ at 5% across epochs 1, 2, 3, and 10.

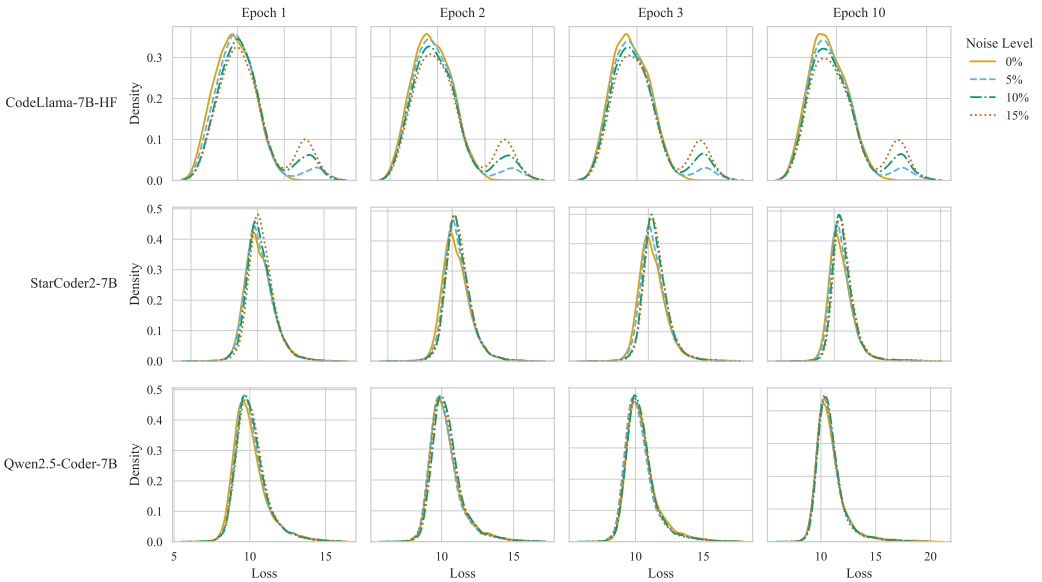


Fig. 3. Loss density distributions by model, epoch, and noise level for code summarization. Each subplot has independent axes.

The plots in Figure 3 provide insights into the behaviors of the evaluated LLMs for code summarization under varying noise levels and training epochs. The clean data is shown with —, while data with 5%, 10%, and 15% inserted noise are shown in - - -, - - -, and - - -, respectively. Firstly, CodeLlama-7B-HF clearly exhibits a dual distribution in its loss values under higher noise conditions (10% and 15%), indicating a distinct separation between clean and noisy samples during training. This separation suggests that CodeLlama-7B-HF effectively learns to distinguish genuine data from corrupted samples, thereby forming a clear boundary in the loss space.

In contrast, StarCoder2-7B and Qwen2.5-Coder-7B show less differentiation in their loss distributions. Both models exhibit a broader tail in higher noise scenarios and peak shifting to the left for the clean data, yet fail to demonstrate clear bi-distributions. This implies a relatively limited capability in discriminating noisy samples based solely on loss values. Interestingly, the performance drop of StarCoder2-7B with noise inserted remains marginal for code summarization (See Table 3), suggesting the presence of robustness within this model to noise interference for this task. Conversely, Qwen2.5-Coder-7B exhibits significant performance degradation under noise conditionss (See Table 3). A plausible explanation for this behaviour is that StarCoder2-7B may inherently has a more robust internal representation, or better learning ability on noisy data, due to its pre-training process, which may include diverse and highly noisy data. This robustness could allow the model to generalize better across different data qualities without the need to explicitly distinguish noisy samples in the loss space.

Finding 3: *Overall, these observations highlight the varying sensitivities of LLMs to noise for code summarization, underscoring the necessity of tailored approaches in noise mitigation.*

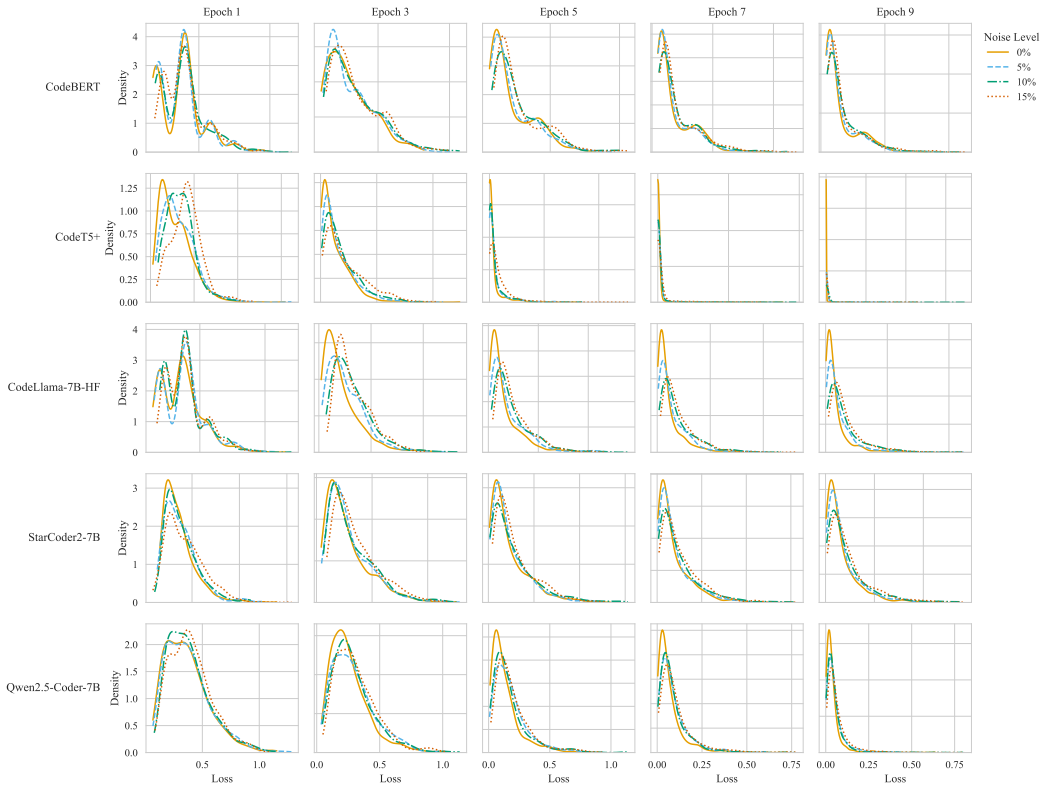


Fig. 4. Loss density distributions by model, epoch, and noise level for code commit intent classification. Each subplot has independent axes.

4.1.2 Code Commit Intent Classification. Figure 4 presents the loss density distributions of the five studied models for commit intent classification, over epochs 1, 3, 5, 7, and 9, with different noise levels: 0 (clean data), 5%, 10%, and 15%. The clean data is shown with —, while data with 5%, 10%, and 15% noise are shown in - - -, - - -, and - - -, respectively. We can see that the higher the noise

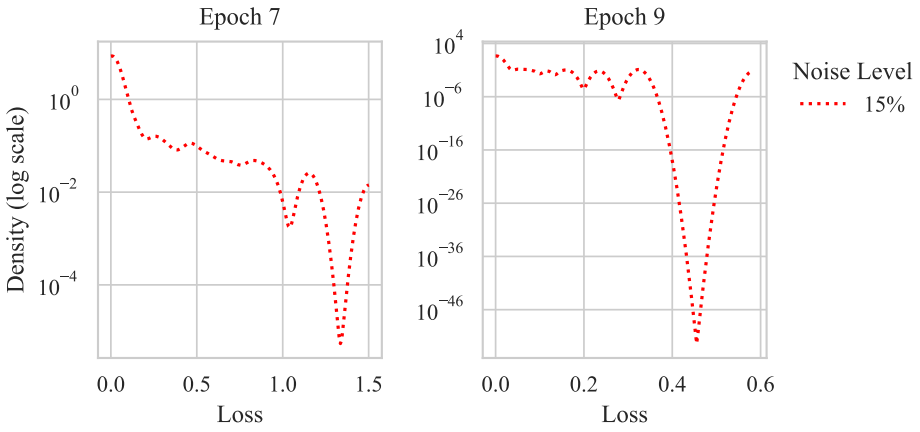


Fig. 5. Loss density distributions for CodeT5+ at epochs 7 and 9 under 15% noise insertion on a log scale.

level, the more difficult it is for the model to fit and converge in each training phase, especially in the last epoch. In epoch 9, the loss distribution of 0% noise samples has been clustered into a very sharp and high peak, meaning that all clean samples are firmly fitted to the same low-loss interval by the model, which is most obvious in CodeT5+, where the height of the peak is so high that other noise levels are almost invisible. As the percentage of noise increases, the corresponding density's peak decreases and becomes progressively wider to the right, with the right tail becoming more pronounced. In other words, it is easier for clean data to find a low-loss solution in the parameter space, while the noise samples can neither share the same loss interval nor be highly aggregated in the distribution due to the conflict with the true distribution. CodeLlama-7B-HF in Figure 4, shows clear noise effects at a very early stage (epoch 3). StarCoder2-7B shows highly consistent convergence after the fifth round of training, regardless of the amount of noise. Even with 15% noise samples, it still demonstrates a wider tail. Qwen2.5-Coder-7B also shows persistent tail thickness, reflecting their strong, but more gradual ability to learn in the presence of noise. These observations mostly align with loss density plots for code summarization (Figures 2a, 2b, and 3), where CodeBERT, CodeT5+and CodeLlama-7B-HF show more clear distinction among the noisy and clean data, while StarCoder2-7Band Qwen2.5-Coder-7B demonstrate less difference.

Finding 4: *Overall, for commit intent classification, all models experience progressively wider tails in later epochs as noise continues to increase, suggesting that the models have difficulty converging the noisy samples alongside the clean ones.*

4.2 Performance of MANTRA

4.2.1 Code Summarization. Table 3 reports the BLEU-4 score for all models studied in code summarization. CodeBERTperforms best with a BLEU-4 of 18.24 on the clean data, but drops to 17.69 at 15% noise, a drop of 0.55. With MANTRA, the score rises to 18.30 at no noise, and more importantly, drops only to 18.14 at 15% noise, narrowing the drop to 0.16. CodeT5+ dropped from 17.71 to 16.86 (down 0.85) under fully fine-tuning, instead slightly increased the score to 18.39 at 5% noise and kept it at 17.77 at 15% noise with MANTRA, a drop of only 0.24 from the clean column. For CodeLlama-7B-HF, the BLEU-4 at 15% noise dropped directly from 14.82 to 11.45, which is an alarming drop; however, with MANTRA, only drops from 12.35 to 11.24, which is a drop of 1.11. The drop of StarCoder2-7B is mild (14.66 to 13.60, a drop of 1.06), but after adding MANTRA, it

not only rises back to 14.52 and 14.60 at 5% and 10% noise, respectively, but also increases to 14.52 and 14.61 at 5% and 10% noise, and a similar performance of 14.49 with 15% noise using MANTRA. Qwen2.5-Coder-7B is the most sensitive model to noise, where the score plummets to 2.66 at 15% noise, but with MANTRA, it recovers to 9.51 at worst, which is a narrower drop and significantly improves the robustness.

The general observation comparing the models' performance with and without noise is similar to RQ1; the higher the proportion of noise, the greater the decline, which is different for various models. We observe a more significant decline in Qwen2.5-Coder-7B.

Table 3. BLEU scores for code summarization (CS) and F1 scores for commit intent classification (F1) under varying noise levels (0%, 5%, 10%, 15%), comparing baseline and MANTRA-enhanced methods.

Model (Method)	0%		5%		10%		15%	
	CS	**F1**	**CS**	**F1**	**CS**	**F1**	**CS**	**F1**
CodeBERT								
full FT	18.24	69.4%	18.19	64.7%	18.01	49.5%	17.69	56.8%
MANTRA	18.30	67.9%	18.21	66.1%	18.18	56.9%	18.14	56.9%
CodeT5+								
full FT	17.71	68.2%	17.25	66.4%	17.05	60.6%	16.86	61.3%
MANTRA	17.51	67.9%	18.39	69.3%	18.13	67.6%	17.77	67.5%
CodeLlama-7B								
LoRA	14.82	68.8%	12.51	65.0%	11.48	60.0%	11.45	59.6%
MANTRA	12.35	72.0%	11.77	76.4%	11.36	72.0%	11.24	69.2%
StarCoder2-7B								
LoRA	14.66	68.5%	14.03	70.3%	13.84	62.4%	13.60	58.6%
MANTRA	14.46	64.2%	14.52	63.2%	14.61	65.1%	14.49	64.8%
Qwen2.5-Coder-7B								
LoRA	10.94	58.7%	6.66	51.6%	3.02	51.3%	2.66	44.8%
MANTRA	12.01	61.1%	10.76	55.1%	10.45	55.3%	9.51	52.3%

4.2.2 Code Commit Intent Classification. Table 3 presents the F1 scores for all models with and without MANTRA. The F1 scores of all models drop significantly as the level of noise increases from to 15%. Most models' scores drop by 8-15 percentage points at 15% noise compared to 0% noise. We see the same trend in our results in the loss density distribution (RQ1). The noisy samples broaden the distribution and lower the peaks, making it difficult for the models to converge to the same low-loss interval as the clean samples.

When comparing the models, CodeBERT achieves the best F1 of 69.4 without noise, but falls to 56.8 with 15% noise, a drop of 12.6 points. Adding MANTRA narrows the drop to 11.0 points. CodeT5+'s performance is much smoother: without MANTRA, the drop from 0% to 15% is 7.6 points (from 68.2 to 60.6), while with MANTRA, the gap almost disappears, with a drop of only 0.4 points (from 67.9 to 67.5), which suggests that in the moderately noisy scenarios, MANTRA significantly improves the robustness of the model to noise. On the other hand, CodeLlama-7B-HF shows excellent robustness: without MANTRA, it drops 9.2 points (from 68.8 to 59.6), while with MANTRA, it drops only 2.8 points (from 72.0 to 69.2). Even with 15% noise level, the performance is better than the 0% data, which improves the overall performance and reduces the performance fluctuation caused by noise.

For StarCoder2-7B, the F1 drops from 68.5 to 58.6 without MANTRA at 15%, but with MANTRA, the performance is improved a bit, being 64.8 under 15% noise. Qwen2.5-Coder-7B, is the most sensitive to noise. Without MANTRA, it drops by 13.9 points (from 58.7 to 44.8). With MANTRA, the drop is reduced to 9.4 points (from 61.1 to 52.3), showing that even though the model's ability to converge to noise is relatively weak, MANTRA still improves its performance and stability under noisy environments.

In summary, the results show high noise hinders convergence, but MANTRA's dynamic removal of noisy samples yields smoother, more consistent performance across all noise levels. CodeT5+ and CodeLlama-7B-HF maintain almost zero degradation after the dropout, Qwen2.5-Coder-7B is still stable in strong noise, and even in CodeBERT , we observe that the performance drop caused by the noise reduces significantly due to the introduction of MANTRA. These results not only validate the effect of noise on convergence that we see in the loss density distribution but also show that, in practice, in order to cope with noisy datasets, MANTRA is a reasonable choice that can significantly improve the stability of the training of the models.

5 DISCUSSION

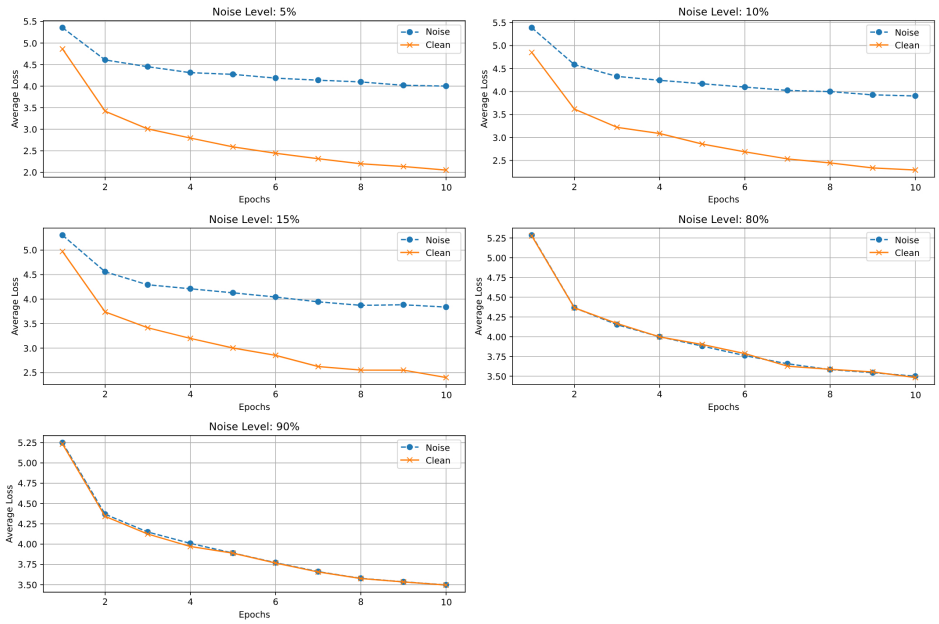


Fig. 6. Loss Across Epochs (Subplots) for CodeBERT

In Figures 6 and 7 we show the difference between the loss of noise sample and clean sample with the discussed noise level, plus two more: 80% and 90% for CodeBERT and CodeT5+, where the common feature of the training dynamics comparing the noise and clean samples can be seen. At the end of the first few epochs, the model quickly pulls down the average loss of most of the clean samples to a very low level, while processing the noise samples slowly, resulting in a relatively flat region where the loss for noise is stable. Taking 5% and 10% noise as an example, both models show that after the second round, the loss of clean samples drops to a very low value, and continues to drop slightly, while the noise samples stay at a high value for several epochs, and also continue

to drop slightly, as seen in Figures 6 and 7. This observation is important because it shows that the model will try to learn from the noise, so the more we train on the noise, the more it damages the performance. This 'fast-then-slow' separation pattern is the resonance effect of the conflict between the noise labels and the true distribution. The model quickly fits the dominant pattern of the clean data on the one hand, and is dragged down by the false signal of the noise samples on the other. Despite the numerical differences, CodeT5+, with its architecture and parameters, converges on the majority of the samples in the second round, whereas CodeBERT requires more epochs to see improvement on the clean samples. Still, the phenomenon of delayed convergence of the noisy samples is shown in both models. A higher proportion of noise raises the noise curve overall. As training progresses, the gap between the two noise levels at the final plateau narrows, suggesting that regardless of the proportion of noise, the model attempts to agree on these samples in a compromise loss region if training is sufficiently deep. However, this plateau is much higher than the minimum loss of the clean samples. This leads to varying degrees of decline in the model's performance for the test set.

This cross-model consistency reveals an important conclusion: *label noise has a constant effect on the convergence of large-scale Transformer models.* Not only does it cause separation in the early stages of training, but it also constantly misleads the model away from the correct pattern. While there are slight differences across model, this separation in noise and clean data trajectory cuts across model types, highlighting the impact of noise on training stability and final performance across tasks and models.

When we look at LLMs for the code commit intention classification, another interesting point shows. Figure 8 shows the average loss of clean versus noisy samples over nine epochs for three LLMs for commit intent classification at noise levels of 5%, 10%, and 15%. In the case of CodeLlama-7B-HF (top row), we can see that the clean sample loss decays smoothly over all epochs, regardless of noise level, whereas the noisy sample loss behaves quite differently. At 5% and 10%

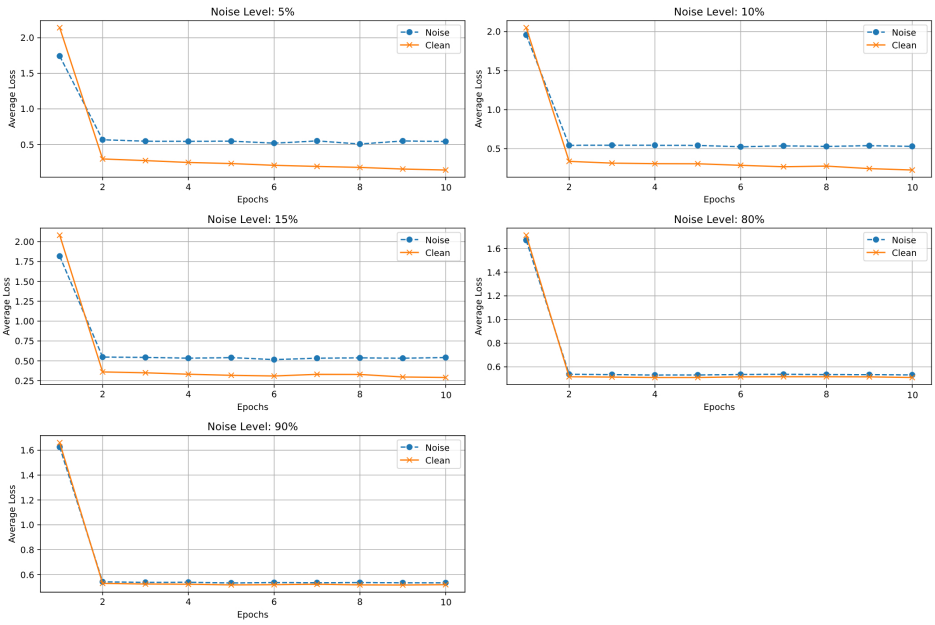


Fig. 7. Loss Across Epochs (Subplots) for CodeT5+

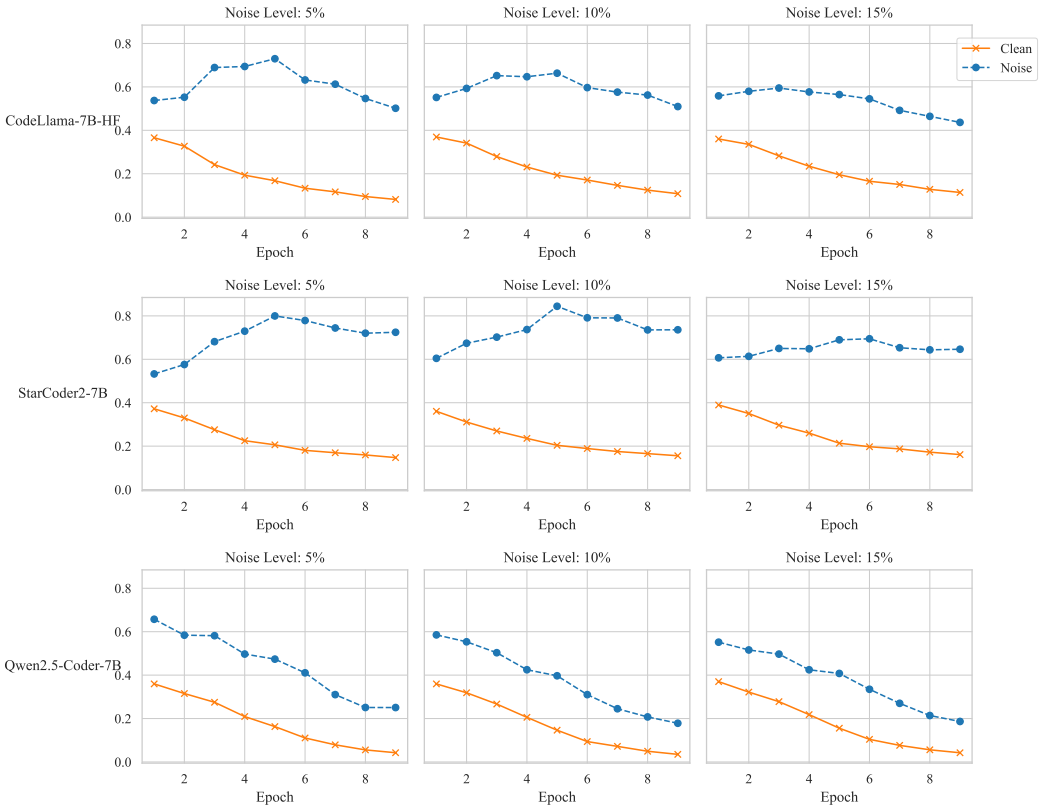


Fig. 8. Average loss of clean versus noisy samples over epochs for three LLMs for commit intent classification at noise levels of 5%, 10%, and 15%. Each row corresponds to one model and each column to one noise level; the solid (×) curve tracks clean-sample loss and the dashed (o) curve tracks noisy-sample loss.

noise, the noisy-loss curve dips slightly, peaks around epoch 5, then declines, signaling that the model initially rejects corrupted labels by amplifying their loss before learning to separate them. At 15% noise, that peak vanishes, and the noisy loss simply falls in lockstep with the clean loss. This implies that the sheer volume of spurious labels overwhelms the model's ability to isolate them. Similar trend can be found in StarCoder2-7B, we see a similar two-phase trajectory but with an even more pronounced peak in noisy sample loss between epochs two and five. In particular, at ten percent noise, the model amplifies error on the corrupted examples to a greater extent than CodeLlama-7B-HF, suggesting a stronger internal mechanism for flagging anomalous data. Yet once the noise ratio increases to fifteen percent, the height of that peak diminishes, suggesting that as noise levels rise beyond certain thresholds, models may struggle to maintain a clear distinction between clean and corrupted samples during fine-tuning.

In contrast, Qwen2.5-Coder-7B (bottom row) shows no such transient resistance. Its noisy sample loss plummets from the first epoch and remains on a steady downward trend, closely tracking the clean sample loss, albeit at a consistently higher level. Even as noise increases, there is no pronounced rise or plateau, no evidence of the model disagreeing with mislabeled examples. This is also reflected in its F1 score: pure LoRA falls to 44.8% at 15% noise, whereas with MANTRA it holds steady at 52.3%, the drop being narrowed considerably, and the F1 score is higher than pure

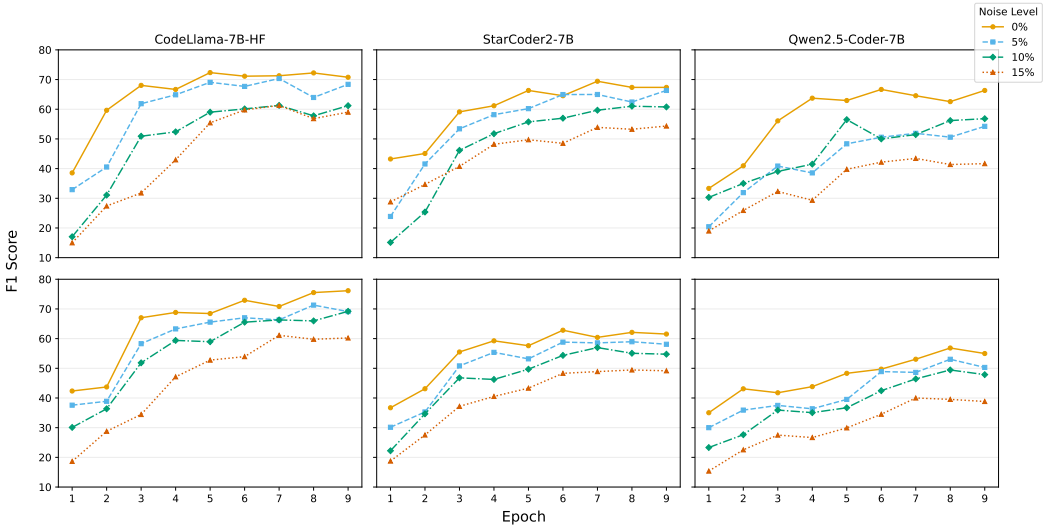


Fig. 9. F1 score across epochs for three LLMs under four label-noise levels. The top row shows LoRA fine-tuning ("Without MANTRA"); the bottom row applies MANTRA.

LoRA at all noise levels. The performance of Qwen2.5-Coder-7B highlights the universal value of MANTRA, which shows that dynamic sample removing not only reduces the interference of mislabeling, but also improves the robustness of the model that is sensitive to noise.

Overall, label noise broadens the loss distribution and slows convergence, degrading performance as noise increases, whereas MANTRA flattens the performance curves and even lets StarCoder2-7B reclaim its lead under certain noise conditions. Models' initial noise resistance and their reactions to dynamic culling reflect differences in capacity, pre-training data richness, and architecture. Future work can tailor culling thresholds and schedules per architecture to optimize performance across noisy scenarios.

Figure 9 shows F1 score across epochs for three LLMs under four label-noise levels. Across all LLMs, the F1 score on the validation set after each epoch without MANTRA (top row) exhibits clear signs of disruption when different levels of noise are inserted into the training set. In the clean setting (0% noise), each model climbs steadily toward its peak F1 and stays stable when it reaches the peak. After injecting noise, however, the ascent breaks into a series of bumps and dips. At 5% noise, the curves still eventually recover, but their peaks come later and they wobble more: CodeLlama-7B-HF's F1 falters around epoch 3 and again at epoch 8, StarCoder2-7B even briefly plateaus or dips under clean performance, and Qwen2.5-Coder-7Bś 10% noise curve spikes irregularly around epoch 5 before settling. As noise rises to 10% and 15%, these oscillations grow more pronounced and the maximum F1 shrinks. This behavior reflects the model oscillating between fitting the majority of clean examples and over-memorizing the noise; it loses ground on true validation patterns when trying to fit to noise, then regains it when the clean signal briefly dominates.

By contrast, once MANTRA's dynamic filtering is applied (bottom row), all three models' F1 trajectories realign almost perfectly with the noise-free baseline. Even at 15% noise, we now see a smooth upward trend that closely shadows the 0% curve, with only a small, consistent gap in absolute F1. The jagged jumps disappear, the peaks return to the same epochs as the clean data, and the models regain confidence much earlier in training. In effect, MANTRA prevents the model

from getting lost in corrupted labels by removing high-loss examples so that each optimization step reinforces genuine patterns, not spurious ones.

Noisy labels induce high-loss examples that pull gradients in opposing directions, causing oscillations in validation loss and F1. By using a GMM to prune those outliers, MANTRA ensures each epoch's updates mirror the clean-data distribution, yielding smooth, convergent F1 curves at any noise level.

Comparing with existing works. Both our work and Shah et al. [42] observe that label noise leaves a clear impact in training dynamics: clean samples rapidly converge to low loss while noisy samples lag, producing broader tails or multiple loss distributions and delayed convergence. Shah et al. [42] report near-zero biases in many layers and significant gradient instability when training on buggy data. In our loss density analysis, we similarly see a pronounced fast then slow separation, where clean losses plummet early, noisy losses plateau higher and much later, which underlies the instability Shah et al. [42] document. Consistent with Wang et al. [45], we confirm that smaller models suffer severe performance degradation even under low noise rates (5–10%). These PTMs lose accuracy quickly as label corruption increases. Whereas Wang et al. [45] find that larger models remain largely robust to label noise, our experiments show that Qwen-2.5-Coder is notably sensitive. It's F1 on the validation set plummets sharply, even at moderate noise (10%), and exhibits pronounced fluctuations across epochs. In contrast, CodeLlama and StarCoder2 still display the resilience Wang et al. [45] describe. Our experiments show that embedding MANTRA during training produces smoother convergence and higher accuracy for different model sizes and architectures. For researchers, this underscores the value of dynamic, model-aware noise mitigation strategies.

6 THREATS TO VALIDITY

Internal Validity. Our experiments introduce noise by randomly flipping ground-truth labels uniformly. While this provides precise control and reproducibility, real-world annotation errors are typically non-uniform—developers show systematic biases, and automated pipelines introduce structured noise patterns. Our synthetic approach may not fully capture these real-world noise characteristics, though we mitigate this by testing multiple noise levels.

External Validity. In our study, we examined two tasks of code summarization and commit intent classification. While our results might not generalize to other tasks, we chose two different types of tasks, one is a generative and another is classification. The variety of task types in our study alleviates the problems that MANTRA could only be applied to certain tasks. Another validity concern relates to the studied datasets. Other datasets and industrial code bases might have different data and noise distribution, deviating from the datasets we examined and the simulated noise. Though we selected widely-used benchmarks, enhancing comparability and relevance across research, the efficacy of MANTRA may need recalibration. That being said, our approach is not specific to a tasks or model, therefore, we anticipate low efforts of applying MANTRA on new tasks and datasets.

Hardware limitation We have targeted five popular Transformer models up to 7 B parameters. However, due to hardware limitations, we have to use LoRA and quantization during fine-tuning, which freezes the vast majority of the pretrained weights. While LoRA has shown results in some experiments that are similar to or even exceed those of full fine-tuning [19], full-parameter fine-tuning could either amplify noise memorization, making MANTRA even more necessary, or conversely absorb noise more smoothly, reducing the relative benefit of our approach.

7 CONCLUSION

In this study, we examined how controlled label noise at 0%, 5%, 10%, and 15% alters the behavior of five code models on code summarisation and commit-intent classification. Noise broadened the loss landscape, postponed the convergence of corrupted examples, and reduced task accuracy. To counteract this effect we introduced MANTRA, a Gaussian-mixture filter that operates during fine-tuning, whether full or LoRA, and is paired with dropout. MANTRA restored a clear separation between clean and noisy samples: at the highest noise level, CodeLlama and CodeT5+ lost under three per cent BLEU or F1, StarCoder recovered its lead, and even the most fragile model, Qwen, regained more than half of its lost accuracy while dropout kept it from over-fitting spurious patterns. The study shows that label noise invariably splits the learning dynamics of clean and corrupted data, that sequence-to-sequence architectures first resist and then assimilate noise, and that adaptive filtering with light regularisation can almost remove the performance penalty imposed by severe corruption.

Future work will refine mixture thresholds, explore alternative filters, adjust dropout schedules, and apply the method to naturally noisy corpora such as issue trackers and pull-request discussions.

REFERENCES

[1] Toufique Ahmed and Premkumar Devanbu. 2022. Multilingual Training for Software Engineering. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1443–1455. https://doi.org/10.1145/3510003.3510049

[2] Yingbin Bai, Erkun Yang, Bo Han, Yanhua Yang, Jiatong Li, Yinian Mao, Gang Niu, and Tongliang Liu. 2021. Understanding and Improving Early Stopping for Learning with Noisy Labels. *CoRR* abs/2106.15853 (2021). arXiv:2106.15853 https://arxiv.org/abs/2106.15853

[3] Aaditya Bhatia, Dayi Lin, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E. Hassan. 2024. Data Quality Antipatterns for Software Analytics. arXiv:2408.12560 [cs.SE] https://arxiv.org/abs/2408.12560

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS'20)*. Curran Associates Inc., Red Hook, NY, USA, Article 159, 25 pages.

[5] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.* 15, 3, Article 39 (March 2024), 45 pages. https://doi.org/10.1145/3641289

[6] Laxmi Narayana Chejarla. 2025. AI Advancements in the TMT Industry: Navigating the Challenges and Business Adaptations. *Journal of Computer Science and Technology Studies* 7, 6 (Jun. 2025), 999–1007. https://doi.org/10.32996/jcsts.2025.7.118

[7] Roland Croft, M. Ali Babar, and Huaming Chen. 2022. Noisy Label Learning for Security Defects. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. 435–447. https://doi.org/10.1145/3524842.3528446

[8] Giuseppe Crupi, Rosalia Tufano, Alejandro Velasco, Antonio Mastropaolo, Denys Poshyvanyk, and Gabriele Bavota. 2025. On the Effectiveness of LLM-as-a-judge for Code Generation and Summarization. *IEEE Transactions on Software Engineering* (2025), 1–17. https://doi.org/10.1109/TSE.2025.3586082

[9] Anh T. V. Dau, Nghi D. Q. Bui, Thang Nguyen-Duc, and Hoang Thanh-Tung. 2022. Towards Using Data-Influence Methods to Detect Noisy Samples in Source Code Corpora. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22)*. ACM, 1–3. https://doi.org/10.1145/3551349.3561168

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[11] Erik Englesson, Amir Mehrpanah, and Hossein Azizpour. 2023. Logistic-Normal Likelihoods for Heteroscedastic Label Noise in Classification. *ArXiv* abs/2304.02849 (2023). https://api.semanticscholar.org/CorpusID:257985149

[12] Siyuan Feng, Jiawei Liu, Ruihang Lai, Charlie Ruan, Yong Yu, Lingming Zhang, and Tianqi Chen. 2025. Productively Deploying Emerging Models on Emerging Platforms: A Top-Down Approach for Testing and Debugging. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA080 (June 2025), 23 pages. https://doi.org/10.1145/3728957

[13] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. https://doi.org/10.48550/ARXIV.2002.08155

[14] Remi Gribonval, Antoine Chatalic, Nicolas Keriven, Vincent Schellekens, Laurent Jacques, and Philip Schniter. 2021. Sketching Data Sets for Large-Scale Learning: Keeping only what you need. *IEEE Signal Processing Magazine* 38, 5 (2021), 12–36. https://doi.org/10.1109/MSP.2021.3092574

[15] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio Cesar, Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks Are All You Need. https://www.microsoft.com/en-us/research/publication/textbooks-are-all-you-need/

[16] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. https://doi.org/10.48550/ARXIV.2009.08366

[17] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor W. Tsang, and Masashi Sugiyama. 2018. Co-teaching: robust training of deep neural networks with extremely noisy labels. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 8536–8546.

[18] Tjaša Heričko, Boštjan Šumak, and Sašo Karakatič. 2024. Commit-Level Software Change Intent Classification Using a Pre-Trained Transformer-Based Code Model. *Mathematics* 12, 7 (2024). https://doi.org/10.3390/math12071012

[19] J. Edward Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. *ArXiv* abs/2106.09685 (2021). https://api.semanticscholar.org/CorpusID:235458009

[20] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Shanghaoran Quan, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder Technical Report. *ArXiv* abs/2409.12186 (2024). https://api.semanticscholar.org/CorpusID:272707390

[21] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[22] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.* 55, 12, Article 248 (March 2023), 38 pages. https://doi.org/10.1145/3571730

[23] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2025. A Survey on Large Language Models for Code Generation. *ACM Trans. Softw. Eng. Methodol.* (July 2025). https://doi.org/10.1145/3747588 Just Accepted.

[24] Zhimeng Jiang, Kaixiong Zhou, Zirui Liu, Li Li, Rui Chen, Soo-Hyun Choi, and Xia Hu. 2022. An Information Fusion Approach to Learning with Instance-Dependent Label Noise. In *International Conference on Learning Representations*. https://api.semanticscholar.org/CorpusID:251648853

[25] Heewon Kim, Hyun Sung Chang, Kiho Cho, Jaeyun Lee, and Bohyung Han. 2024. Learning with Noisy Labels: Interconnection of Two Expectation-Maximizations. *ArXiv* abs/2401.04390 (2024). https://api.semanticscholar.org/CorpusID:266900098

[26] Seong Min Kye, Kwanghee Choi, Joonyoung Yi, and Buru Chang. 2021. Learning with Noisy Labels by Efficient Transition Matrix Estimation to Combat Label Miscorrection. *ArXiv* abs/2111.14932 (2021). https://api.semanticscholar.org/CorpusID:244729337

[27] Jake Lever, Sibo Cheng, César Quilodrán Casas, Che Liu, Hongwei Fan, Robert Platt, Andrianirina Rakotoharisoa, Eleda Johnson, Siyi Li, Zhendan Shang, and Rossella Arcucci. 2025. Facing and mitigating common challenges when working with real-world data: The Data Learning Paradigm. *Journal of Computational Science* 85 (2025), 102523. https://doi.org/10.1016/j.jocs.2024.102523

[28] Junnan Li, Richard Socher, and Steven C. H. Hoi. 2020. DivideMix: Learning with Noisy Labels as Semi-supervised Learning. *ArXiv* abs/2002.07394 (2020). https://api.semanticscholar.org/CorpusID:211146562

[29] Zhong Li, Minxue Pan, Yu Pei, Tian Zhang, Linzhang Wang, and Xuandong Li. 2022. Robust learning of deep predictive models from noisy and imbalanced software engineering datasets. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[30] Sheng Liu, Jonathan Niles-Weed, Narges Razavian, and Carlos Fernandez-Granda. 2020. Early-Learning Regularization Prevents Memorization of Noisy Labels. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 20331–20342. https://proceedings.neurips.cc/paper_files/paper/2020/file/ea89621bee7c88b2c5be6681c8ef4906-Paper.pdf

[31] Yang Liu and Hongyi Guo. 2020. Peer loss functions: learning from noisy labels without knowing noise rates. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 578, 11 pages.

[32] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan L. Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, W. Yu, Lucas Krauss, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alexander Gu, Binyuan Hui, Tri Dao, Armel Randy Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian J. McAuley, Han Hu, Torsten Scholak, Sébastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. StarCoder 2 and The Stack v2: The Next Generation. *ArXiv* abs/2402.19173 (2024). https://api.semanticscholar.org/CorpusID:268063676

[33] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. https://doi.org/10.48550/ARXIV.2102.04664

[34] Junyu Luo, Xiao Luo, Kaize Ding, Jingyang Yuan, Zhiping Xiao, and Ming Zhang. 2024. RobustFT: Robust Supervised Fine-tuning for Large Language Models under Noisy Response. arXiv:2412.14922 [cs.CL] https://arxiv.org/abs/2412.14922

[35] Wenqiang Luo, Jacky Keung, Boyang Yang, He Ye, Claire Le Goues, Tegawendé F. Bissyandé, Haoye Tian, and Xuan Bach D. Le. 2025. When Fine-Tuning LLMs Meets Data Privacy: An Empirical Study of Federated Learning in LLM-Based Program Repair. *ACM Trans. Softw. Eng. Methodol.* (May 2025). https://doi.org/10.1145/3733599 Just Accepted.

[36] Changan Niu, Chuanyi Li, Vincent Ng, Dongxiao Chen, Jidong Ge, and Bin Luo. 2023. An Empirical Comparison of Pre-Trained Models of Source Code. *arXiv e-prints*, Article arXiv:2302.04026 (Feb. 2023), arXiv:2302.04026 pages. https://doi.org/10.48550/arXiv.2302.04026 arXiv:2302.04026 [cs.SE]

[37] Curtis Northcutt, Lu Jiang, and Isaac Chuang. 2021. Confident Learning: Estimating Uncertainty in Dataset Labels. *J. Artif. Int. Res.* 70 (may 2021), 1373–1411. https://doi.org/10.1613/jair.1.12125

[38] Dan Qiao, Chenchen Dai, Yuyang Ding, Juntao Li, Qiang Chen, Wenliang Chen, and Min Zhang. 2022. SelfMix: Robust Learning against Textual Label Noise with Self-Mixup Training. In *Proceedings of the 29th International Conference on Computational Linguistics*, Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na (Eds.). International Committee on Computational Linguistics, Gyeongju, Republic of Korea, 960–970. https://aclanthology.org/2022.coling-1.80/

[39] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language Models are Unsupervised Multitask Learners. (2018). https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

[40] Daniel Rodriguez-Cardenas, Alejandro Velasco, and Denys Poshyvanyk. 2025. SnipGen: A Mining Repository Framework for Evaluating LLMs for Code. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. 508–512. https://doi.org/10.1109/MSR66628.2025.00084

[41] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[42] Mehil B Shah, Mohammad Masudur Rahman, and Foutse Khomh. 2024. Towards Understanding the Impact of Data Bugs on Deep Learning Models in Software Engineering. *arXiv preprint arXiv:2411.12137* (2024).

[43] Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. 2025. Can LLMs Replace Human Evaluators? An Empirical Study of LLM-as-a-Judge in Software Engineering. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA086 (June 2025), 23 pages. https://doi.org/10.1145/3728963

[44] Song Wang, Chetan Bansal, and Nachiappan Nagappan. 2021. Large-scale intent analysis for identifying large-review-effort code changes. *Information and Software Technology* 130 (2021), 106408. https://doi.org/10.1016/j.infsof.2020.106408

[45] Wenhan Wang, Yanzhou Li, Anran Li, Jian Zhang, Wei Ma, and Yang Liu. 2024. An empirical study on noisy label learning for program understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[46] Yue Wang, Hung Le, Akhilesh Gotmare, Nghi Bui, Junnan Li, and Steven Hoi. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 1069–1088. https://doi.org/10.18653/v1/2023.emnlp-main.68

[47] Zhaokun Wang, Jinyu Guo, Jingwen Pu, Lingfeng Chen, Hongli Pu, Jie Ou, Libo Qin, and Wenhong Tian. 2025. Noise-Robustness Through Noise: Asymmetric LoRA Adaption with Poisoning Expert. arXiv:2505.23868 [cs.LG] https://arxiv.org/abs/2505.23868

[48] Chen Wu, Shikai Guo, Hui Li, Chenchen Li, and Rong Chen. 2025. Estimating Uncertainty in Line-Level Defect Prediction via Perceptual Borderline Oversampling. *ACM Trans. Softw. Eng. Methodol.* (June 2025). https://doi.org/10.1145/3746225 Just Accepted.

[49] Aybüke Yalçıner, Ebru Gökalp, and Ahmet Dikici. 2025. Transforming Software Engineering Processes Through Generative AI: A Framework for Integration and Implementation. *Computer* 58, 7 (2025), 53–65. https://doi.org/10.1109/MC.2025.3539347

[50] Yu Yao, Mingming Gong, Yuxuan Du, Jun Yu, Bo Han, Kun Zhang, and Tongliang Liu. 2023. Which is Better for Learning with Noisy Labels: The Semi-supervised Method or Modeling Label Noise?. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:260815923

[51] Liqing Ye, Agam Shah, Chao Zhang, and Sudheer Chava. 2025. Calibrating Pre-trained Language Classifiers on LLM-generated Noisy Labels via Iterative Refinement. *ArXiv* abs/2505.19675 (2025). https://api.semanticscholar.org/CorpusID:278905085

[52] Chenglin Yu, Xin Ma, and Weiwei Liu. 2023. Delving into Noisy Label Detection with Clean Data. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:260871868

[53] Bo Yuan, Yulin Chen, Yin Zhang, and Wei Jiang. 2024. Hide and Seek in Noise Labels: Noise-Robust Collaborative Active Learning with LLMs-Powered Assistance. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 10977–11011. https://doi.org/10.18653/v1/2024.acl-long.592

[54] Peng-Fei Zhang, Zi Huang, Xin Luo, and Pengfei Zhao. 2022. Robust Learning with Adversarial Perturbations and Label Noise: A Two-Pronged Defense Approach. In *Proceedings of the 4th ACM International Conference on Multimedia in Asia* (<conf-loc>, <city>Tokyo</city>, <country>Japan</country>, </conf-loc>) *(MMAsia '22)*. Association for Computing Machinery, New York, NY, USA, Article 23, 7 pages. https://doi.org/10.1145/3551626.3564934

[55] Shuo Zhang, Yuwen Li, Zhongyu Wang, Jianqing Li, and Chengyu Liu. 2024. Learning with Noisy Labels Using Hyperspherical Margin Weighting. In *AAAI Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:268701613

[56] Yuxiang Zhu and Minxue Pan. 2019. Automatic Code Summarization: A Systematic Literature Review. *ArXiv* abs/1909.04352 (2019).

[57] Zhaowei Zhu, Zihao Dong, and Yang Liu. 2021. Detecting Corrupted Labels Without Training a Model to Predict. In *International Conference on Machine Learning*. https://api.semanticscholar.org/CorpusID:246431058

Received 01 October 2024; revised 2024; accepted 10 April 2024