Efficient Code LLM Training via Distribution-Consistent and Diversity-Aware Data Selection

Anonymous ACL submission

Abstract

Recent advancements in large language models (LLMs) have significantly improved code generation and program comprehension, accelerating the evolution of software engineering. Current methods primarily enhance model performance by leveraging vast amounts of data, focusing on data quantity while often overlooking data quality, thereby reducing training efficiency. To address this, we introduce an approach that utilizes a parametric model for code data selection, aimed at improving both training efficiency and model performance. Our method optimizes the parametric model to ensure distribution consistency and diversity within the selected subset, guaranteeing high-quality data. Experimental results demonstrate that using only 10K samples, our method achieves gains of 2.4% (HumanEval) and 2.3% (MBPP) over 92K full-sampled baseline, outperforming other sampling approaches in both performance and efficiency. This underscores that our method effectively boosts model performance while significantly reducing computational costs. Code is available at here.

1 Introduction

Recent years have witnessed remarkable progress in large language models (LLMs), particularly in code-related domains (Hurst et al., 2024; Liu et al., 2024a; Guo et al., 2025). Open-source code models (Li et al., 2023b; Lozhkov et al., 2024; Guo et al., 2024) have significantly advanced academic research by demonstrating strong capabilities in code generation and program comprehension. Through large-scale pre-training, these models provide intelligent support across multiple programming languages and development environments, accelerating the evolution of software engineering intelligence. Meanwhile, instruction tuning has proven to be an effective method to enhance model performance (Wei et al., 2021; Chung et al., 2024). By fine-tuning on large-scale instruction data, models better align with human intent and excel in specific tasks. However, high-quality human-annotated data is scarce and costly to obtain. To address this, researchers have proposed various methods for generating data to expand instruction datasets (Wang et al., 2023; Gunasekar et al., 2023).

Recent studies (Zhou et al., 2023a; Xia et al., 2024) emphasize that data quality is more important than quantity, highlighting the significance of representative data selection for training efficiency. To address this, many approaches (Chen et al., 2024; Lu et al., 2024; Wang et al., 2024) leverage advanced LLMs (e.g., ChatGPT) for data selection and annotation, thereby optimizing data quality and diversity. However, these methods face economic challenges when scaling to large datasets, and most algorithms target general tasks rather than code-specific data selection.

Inspired by ActiveFT (Xie et al., 2023), we introduce the parametric model into code data selection to improve training efficiency and model performance. Unlike traditional discrete selection methods, we operate in feature space for more effective data curation. Specifically, we first map each data sample to a high-dimensional feature space using a feature encoder. Then, we construct the parametric model based on these feature representations. Our goal is to ensure that the feature distribution of the selected subset closely matches the original dataset's while maximizing the diversity within the subset. To achieve this, we continuously optimize the parametric model through a loss function that balances distribution consistency and diversity constraints. Finally, based on the optimized parametric model, we select samples most similar to the parameters to construct a high-quality subset.

We conducted extensive experiments to validate the effectiveness of our method. We mixed multiple datasets to construct a training set containing 92K Python samples and trained the model based on DeepSeek-Coder-Base-6.7B. The results

show that, with only 10K sampled data, our method achieved 69.5% on HumanEval and 77.2% on MBPP, surpassing full-data training by 2.4% and 2.3%, respectively. Additionally, our method outperforms other sampling approaches across various data scales while requiring minimal sampling time. This demonstrates that our approach not only effectively selects high-quality data but also significantly improves model performance and computational efficiency.

The contributions of our work are summarized as follows:

- As far as we know, we are the first to introduce the parametric model into code data selection. By ensuring distribution consistency and diversity, we successfully identify high-quality data, significantly improving model performance.
- We perform data selection in the feature space, avoiding traditional discrete selection methods. This greatly enhances sampling and training efficiency.
- Extensive experiments validate the effectiveness of our method. The results show that using only 10K sampled data outperforms full-data training. Moreover, our method surpasses other sampling methods in both performance and sampling time across various data scales.

2 Related Work

2.1 Code Large Language Models

The emergence of large language models (LLMs) has significantly advanced the intelligence-driven transformation of software engineering, particularly demonstrating breakthrough capabilities in code generation and program comprehension tasks. The open-source community has developed multiple high-performance code LLMs, with notable representatives including CodeLlama (Roziere et al., 2023), DeepSeek-Coder (Guo et al., 2024), CodeGemma (Team et al., 2024), and Qwen2.5-Coder (Hui et al., 2024). CodeLlama, built upon the Llama 2 (Touvron et al., 2023) architecture through continued pre-training on 500B code tokens, achieves performance comparable to commercial models in standardized benchmarks. The DeepSeek-Coder series, trained from scratch on 2T high-quality multilingual code corpora, demonstrates significant superiority over the same-scale counterparts in benchmark evaluations. The Qwen2.5-Coder series are built upon the Qwen2.5 (Yang et al., 2024) architecture and continue training on a vast corpus of over 5.5 trillion tokens, surpassing closed-source models like GPT-4 (Achiam et al., 2023) across multiple benchmarks and establishing new state-of-the-art records. These open-source models, through architectural innovations and training strategy optimizations, not only deliver highly customizable code intelligence solutions but also construct efficient technical infrastructure for both academic research and industrial applications.

2.2 Instruction Fine-tuning

Instruction fine-tuning has been demonstrated as a crucial approach to enhance model performance and align models with human preferences. The study (Chung et al., 2024) indicates that scaling both the number of tasks and the model size through instruction fine-tuning yields performance improvements across various model classes. For instance, WizardCoder (Luo et al., 2023) leverages the Evol-Instruct (Xu et al., 2024) method to iteratively evolve the complexity of the CodeAlpaca (Chaudhary, 2023) dataset. This approach results in a fine-tuning dataset consisting of approximately 78K highly complex programming instructions, thereby improving the performance of CodeLLama-Python-34B to 73.2% on the HumanEval benchmark. Similarly, Magicoder (Wei et al., 2024) proposes the OSS-Instruct approach, which generates highly diverse instruction data by using open-source code snippets. This approach successfully generated a dataset containing approximately 75K entries. Additionally, WaveCoder (Yu et al., 2024) makes full use of open-source code data through a carefully designed generator-discriminator data synthesis framework to generate high quality and diverse instruction data in multi-task scenarios.

2.3 Data Selection for Efficient Training

Instruction fine-tuning typically requires large amounts of data, but research such as LIMA (Zhou et al., 2023a) has pointed out that data quality is more crucial than quantity. Therefore, selecting the most valuable data to improve training efficiency has become a focal point of research. DEITA (Liu et al., 2024b) focuses on the complexity, quality and diversity of the data, designing a multi-faceted approach to select instruction data. Based on the Evol-Instruct technique, ChatGPT is used to augment the instructions, and the instructions

are then evaluated for complexity and quality by specially trained scorers. Quantifiable metrics like PPL (Ankner et al., 2024), IFD (Li et al., 2024b), and Superfiltering (Li et al., 2024a) focus on identifying hard samples that are difficult to learn. DQ (Zhou et al., 2023b) integrates data distillation and coreset selection (Iyer et al., 2021) techniques, emphasizing the selection of diverse data. Some methods (Chen et al., 2024; Lu et al., 2024; Xu et al., 2023) that rely on external oracles use powerful language models, such as ChatGPT, for data selection. However, due to cost constraints, utilizing external oracles is not always feasible.

Overall, many data selection algorithms are primarily designed for general tasks, and the integration of parametric models into code data selection remains underexplored.

3 Methodology

Our methodology aims to identify and select high-quality, representative data samples so that training on the curated subset yields better performance than training on the entire dataset. Inspired by ActiveFT (Xie et al., 2023), we incorporate the parametric model into the code data selection process to improve both training efficiency and model performance. We start by defining the data selection task in Section 3.1. Then we introduce the integration of parametric models into data selection in Section 3.2, as illustrated in Figure 1. Finally, the implementation details are provided in Section 3.3.

3.1 Task Definition

Given a large instruction tuning dataset $D = \{x_1, x_2, \ldots, x_n\}$, where each $x_i = (I_i, C_i)$ represents an individual instruction-code pair, our goal is to select a subset $S_\pi^m \subset D$ of size m using a selection strategy π. The performance of the model after fine-tuning on S_π^m is denoted by $P(S_\pi^m)$ and is used to evaluate the effectiveness of the selected subset. The optimal strategy π^* under a fixed budget m is defined as:

$$\pi^* = \arg\max_{\pi} P(S_\pi^m). \quad (1)$$

3.2 Data Selection with Parametric Model

During data selection, we consider two key factors: On one hand, the selected subset S should have the distribution as close as possible to that of the original dataset D. On the other hand, the subset S should maintain the diversity. By balancing these two aspects, we aim to select a representative subset that covers edge cases in the original dataset, thereby enhancing model performance.

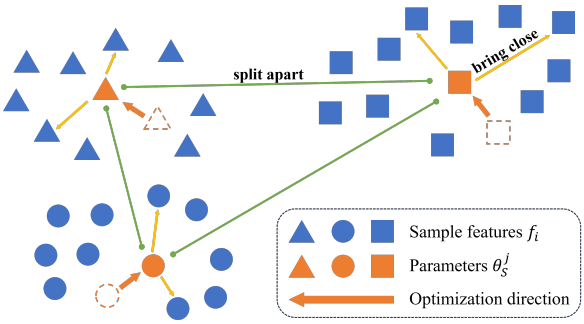

Figure 1: **Parametric Model Optimization Process:** By optimizing the loss in Equation 7, each parameter θ_S^j is attracted to nearby sample features(gold in the figure, Equation 5) and repelled by other parameters $\theta_S^k, k \neq j$ (green in the figure, Equation 6).

Compared to selecting in the discrete data space, it is more efficient and feasible to conduct data selection in the feature space. We map each data sample x_i to the high-dimensional feature space using a feature encoder E, resulting in $f_i = E(x_i)$ which is then normalized. After normalization, we obtain the feature set of the dataset D as $F_D = \{f_i\}_{i\in[n]}$, whose distribution is denoted by p_{F_D}.

Similarly, for the selected subset S, we define its feature set as $F_S = \{f_j\}_{j\in[m]}$, with its corresponding distribution p_{F_S}. Our objective is to find the optimal selection strategy π^* as follows:

$$\pi^* = \arg\min_{\pi} \left[M(p_{F_D}, p_{F_S}) - \lambda \cdot R(S_\pi^m)\right], \quad (2)$$

where $M(\cdot,\cdot)$ measures the distance between two feature distributions, $R(\cdot)$ evaluates the diversity of the selected subset, and λ is a scaling factor that balances two terms. The first term in Equation 2 aims to align the distributions, while the second term ensures the diversity within the subset.

It's challenging to directly optimize the selection strategy π in the discrete space, so we adopt a parametric model p_{θ_S} to approximate p_{F_S}, where $\theta_S = \{\theta_S^j\}_{j\in[m]}$ are the continuous parameters. Each optimized parameter θ_S^j corresponds to the feature of a selected sample f_j, and we select the feature f_j that is closest to θ_S^j. Thus, the optimization objective is written as follows:

$$\theta_{S,\pi^*} = \arg\min_{\theta_S} \left[M(p_{F_D}, p_{\theta_S}) - \lambda \cdot R(\theta_S)\right] \text{ s.t. } ||\theta_S^j||_2 = 1. \quad (3)$$

The key distinction between the features $F_S = \{f_j\}$ and the parameters $\theta_S = \{\theta_S^j\}$ is that f_j is a discrete feature corresponding to a data sample, whereas θ_S^j is continuous in the feature space. By optimizing θ_S, we search for ideal samples in the feature space that cover the original data distribution while maintaining dispersion.

The first term represents the distribution matching term $M(p_{F_D}, p_{\theta_S})$, which measures the similarity between the feature distribution of the dataset and the parametric model. It is defined as:

$$c_i = \arg\max_{j \in [m]} sim(f_i, \theta_S^j), \tag{4}$$

$$M(p_{F_D}, p_{\theta_S}) = - \mathop{E}_{f_i \in F_D} \left[sim(f_i, \theta_S^{c_i})/\tau \right], \tag{5}$$

where $sim(\cdot, \cdot)$ is a similarity function, such as cosine similarity, and τ is a temperature parameter that controls the sensitivity of similarity measure. This term encourages the parameters to be well-spread across the feature space, effectively covering the distribution of the original features.

The second term is the diversity regularization $R(\theta_S)$, which promotes diversity within the selected subset by minimizing the similarity between the selected samples. It is defined as:

$$R(\theta_S) = - \mathop{E}_{j \in [m]} \left[\log \sum_{k \neq j, k \in [m]} \exp\left(sim(\theta_S^j, \theta_S^k)/\tau \right) \right]. \tag{6}$$

We jointly optimize the following loss function to achieve the goal in Equation 3, where λ is set to 1 by default.

$$\begin{aligned} L =& M(p_{F_D}, p_{\theta_S}) - \lambda \cdot R(\theta_S) \\ =& - \mathop{E}_{f_i \in F_D} \left[\mathrm{sim}(f_i, \theta_S^{c_i})/\tau \right] \\ &+ \mathop{E}_{j \in [m]} \left[\log \sum_{k \neq j, k \in [m]} \exp\left(\mathrm{sim}(\theta_S^j, \theta_S^k)/\tau \right) \right]. \end{aligned} \tag{7}$$

We optimize the loss function in Equation 7 using gradient descent. Upon completion of the optimization, we find the features $\{f_j\}_{j \in [m]}$ that exhibit the highest similarity to θ_S^j.

$$f_j = \arg\max_{f_i \in F_D} \mathrm{sim}(f_i, \theta_S^j). \tag{8}$$

Finally, we collect the data samples corresponding to these selected features to form the instruction subset S_π^m, which will be used for fine-tuning.

Algorithm 1: Pseudo-code for Our Data Selection Algorithm

Input: Dataset $D = \{x_i\}_{i \in [n]}$, feature encoder E, selection budget m, iteration number T for optimization
Output: Subset $S = \{x_j\}_{j \in [m]}$

for $i \in [n]$ **do**
  $f_i = E(x_i)$;
// Construct $F_D = \{f_i\}_{i \in [n]}$ based on D, normalized to $||f_i||_2 = 1$
Randomly sample $\{f_j^0\}_{j \in [m]}$ from F_D, and initialize $\theta_S^j = f_j^0$;
// Initialize the parameters $\theta_S = \{\theta_S^j\}_{j \in [m]}$
for $iter \in [T]$ **do**
  Calculate the similarity between $\{f_i\}_{i \in [n]}$ and $\{\theta_S^j\}_{j \in [m]}$: $Sim_{i,j} = f_i^\top \theta_S^j / \tau$;
  $MaxSim_i = \max_{j \in [m]} Sim_{i,j} = Sim_{i,c_i}$;
  // The Top-1 similarity between f_i and θ_S^j, $j \in [m]$
  Calculate the similarity between θ_S^j and $\theta_S^k, k \neq j$ for regularization: $RegSim_{j,k} = \exp({\theta_S^j}^\top \theta_S^k / \tau), k \neq j$;
  $Loss = -\frac{1}{n} \sum_{i \in [n]} MaxSim_i + \frac{1}{m} \sum_{j \in [m]} \log\left(\sum_{k \neq j, k \in [m]} RegSim_{j,k} \right)$;
  // Calculate the loss function in Equation 7
  $\theta_S = \theta_S - lr \cdot \nabla_{\theta_S} Loss$;
  // Optimize the parameter through gradient descent
  $\theta_S^j = \theta_S^j / ||\theta_S^j||_2, j \in [m]$;
  // Normalize the parameters to ensure $||\theta_S^j||_2 = 1$
for $j \in [m]$ **do**
  Find f_j closest to θ_S^j: $f_j = \arg\max_{k \in [n]} f_k^\top \theta_S^j$;
  Find x_j corresponding to f_j;
Return the subset $S = \{x_j\}_{j \in [m]}$;

3.3 Implementation Details

Algorithm 1 illustrates the implementation details of the data selection process. We use the pre-trained model *all-mpnet-base-v2*[1] from the

[1] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

sentence-transformers (Reimers and Gurevych, 2019) library as the feature encoder. This model has been trained on over 1 billion text pairs, effectively capturing the semantic information in instruction texts. For each data sample $x_i = (I_i, C_i)$, we encode the instruction text I_i to obtain the feature vector $f_i = E(I_i) \in \mathbb{R}^{768}$, as the instruction text fully defines the task semantics. All features are then processed using L2 normalization, forming the normalized feature set $F_D = \{f_i\}_{i \in [n]}$.

Before optimizing the parametric model, the parameters θ_S are initialized by randomly sampling from the feature set F_D. During each iteration, to avoid memory overflow, we calculate the similarity between sample features and parameters in a batch-wise manner. Subsequently, we update c_i according to Equation 4. Finally, we compute the loss function in Equation 7 and update the parameters θ_S using gradient descent.

When the optimization process is finished, we find the sample feature f_j that exhibit the highest similarity to θ_S^j according to Equation 8. The corresponding original sample x_j is selected for subsequent fine-tuning. This process ensures that the selected subset is highly representative and covers the edge cases in the original dataset, ultimately providing high-quality data support for the fine-tuning process.

4 Experimental Setup

4.1 Dataset

We use three open-source instruction datasets, including *Evol-Instruct-Python-26K*[2], *CodeExercise-Python-27K*[3], and *OSS-Instruct-75K*[4]. The *Evol-Instruct-Python-26K* dataset is the Python subset of the *Evol-Instruct-80K*[5] dataset, and its construction follows an iterative evolution strategy. Based on the CodeAlpaca (Chaudhary, 2023) instruction dataset, multi-round complexity enhancement operations are applied to the original problems using ChatGPT. The *CodeExercise-Python-27K* dataset is generated using the Camel (Li et al., 2023a) framework, covering hundreds of Python-related topics including basic syntax and data structures, algorithm applications, database queries, machine learning, and more. The *OSS-Instruct-75K* dataset utilizes ChatGPT to generate programming problems and their corresponding solutions. Its uniqueness lies in the use of open-source code snippets as guidance for generation. To maintain consistency in the programming language, we filter this dataset and only retain Python-related entries. Finally, we merge these three datasets to obtain a mixed dataset, *Mix-Python-92K*. We use this mixed dataset for training.

4.2 Evaluation Benchmarks

HumanEval/HumanEval+. HumanEval (Chen et al., 2021) comprises 164 Python programming problems designed to assess the ability of code generation models. Each problem is accompanied by approximately 9.6 test cases to check whether the generated code works as expected. HumanEval has become one of the most widely used benchmarks for evaluating the performance of code LLMs, making it a key tool in the field of artificial intelligence for coding. To enhance the rigor of the evaluation, HumanEval+ (Liu et al., 2023) builds upon the original dataset by significantly increasing the number of test cases through the use of LLMs and mutation strategies, resulting in a more comprehensive evaluation benchmark.

MBPP/MBPP+. MBPP (Austin et al., 2021) consists of approximately 1,000 Python programming challenges sourced from a crowd of contributors, targeting beginners in programming and focusing on core principles and the usage of the standard library. Each challenge includes a description, a solution and three tests to verify the accuracy. To improve the reliability of the benchmark, MBPP+ (Liu et al., 2023) extends the original dataset by incorporating a subset of hand-verified problems from the MBPP-sanitized dataset, ensuring that the tasks are well-defined and unambiguous. This enhances the benchmark's reliability and suitability for more rigorous evaluations.

4.3 Training Details

In our experiments, we use DeepSeekCoder-Base-6.7B as the base model and conduct training on eigth NVIDIA A800-80GB GPUs using PyTorch's Fully Sharded Data Parallel (FSDP) module for 3 epochs. Specifically, we employ the AdamW optimizer with a learning rate of 5e-5, a cosine learning rate scheduler, and 100 warmup steps. The maximum sequence length per batch is set to 4096 tokens with a global batch size of 512. To improve

[2]https://huggingface.co/datasets/mlabonne/Evol-Instruct-Python-26k

[3]https://huggingface.co/datasets/codefuse-ai/CodeExercise-Python-27k

[4]https://huggingface.co/datasets/ise-uiuc/Magicoder-OSS-Instruct-75K

[5]https://huggingface.co/datasets/nickrosh/Evol-Instruct-Code-80k-v1

Methods	Data Size	Sampling Time	HumanEval	HumanEval+	MBPP	MBPP+
Random	10K	-	64.6%	61.0%	74.3%	61.9%
DQ	10K	19.9h	64.6%	60.4%	75.9%	62.4%
DEITA	10K	7.2h	65.2%	61.0%	75.4%	63.0%
PPL	10K	3.6h	62.2%	54.9%	74.6%	61.1%
IFD	10K	3.6h	63.4%	57.3%	62.4%	46.6%
K-Center	10K	42.5 min	64.6%	61.0%	74.9%	61.6%
Ours	10K	**13.5min**	**69.5%**	**63.4%**	**77.2%**	**63.2%**

Table 1: Comparison of different sampling methods. All methods select 10K samples to train DeepSeekCoder-Base-6.7B model. Our method outperforms the others in terms of Pass@1 (%) metric, while reducing sampling time.

training efficiency, we utilize the Dynamic Pack (Lv et al., 2025). Model evaluation is conducted using the EvalPlus (Liu et al., 2023) library.

For optimizing the parametric model, we adopt the same hyperparameter settings as in ActiveFT (Xie et al., 2023). Specifically, we use the Adam optimizer with 300 optimization iterations, a learning rate of 0.001, and set τ in Equation 7 to 0.07.

5 Results

5.1 Sampling Methods

We compared various sampling methods, with results presented in Table 1. when sampling 10K samples, our method outperforms all baseline approaches across all benchmarks. Specifically, our method achieves 69.5% on HumanEval, 63.4% on HumanEval+, 77.2% on MBPP, and 63.2% on MBPP+. These results demonstrate the advantage of co-optimizing for distribution consistency and sample diversity.

K-Center (Sener and Savarese, 2018) and DQ (Zhou et al., 2023b) focus on maximizing diversity within the subset. While these approaches increase inter-sample diversity, they do not guarantee alignment with the original dataset's distribution. Both DQ and K-Center achieve 64.6% on HumanEval, which is lower than our method. This suggests that while diversity is important, it must be balanced with distribution consistency to fully leverage the dataset's structure. PPL and IFD prioritize the selection of complex samples, aiming to enhance model performance by focusing on challenging instances. However, these methods perform poorly compared to random sampling on several benchmarks. For instance, IFD shows a substantial performance drop on MBPP and MBPP+, indicating that complexity-focused sampling may deviate from the original distribution. DEITA combines complexity and quality assessment, yielding competitive results. However, it requires the training of two additional scoring models, resulting in higher computational costs and longer training times. In contrast, our method achieves superior performance with significantly lower computational overhead, making it more efficient and scalable.

5.2 Sampling Efficiency

We compared the sampling efficiency of different methods, with results presented in Table 1. The sampling process can be divided into two main phases: (1) the processing phase, which involves data preprocessing such as feature extraction and model scoring, and (2) the sampling phase, where specific strategies are applied for sample selection.

Among all compared methods, DQ exhibits the lowest sampling efficiency. Its processing phase requires nearly 20 hours to partition the dataset into non-overlapping bins, a duration that even exceeds the model training time. Although DQ achieves competitive performance, its high time cost during the sampling process represents a significant disadvantage. In contrast, the DEITA, PPL, and IFD methods show better efficiency but still require forward inference for each data sample. Specifically, DEITA relies on two 13B-parameter scoring models, while PPL and IFD depend on the model being trained for scoring. As a result, their sampling time increases linearly with the size of the dataset, resulting in poor scalability. The K-Center method reduces sampling time to 42.5 minutes by iteratively computing the similarity between candidate samples and the selected subset. However, its reliance on distance-based computations fundamentally limits further efficiency improvements. In contrast, our method achieves the highest sampling efficiency, requiring only 13.5 minutes. This advantage is attributed to the learnable parametric model for data selection, which significantly reduces com-

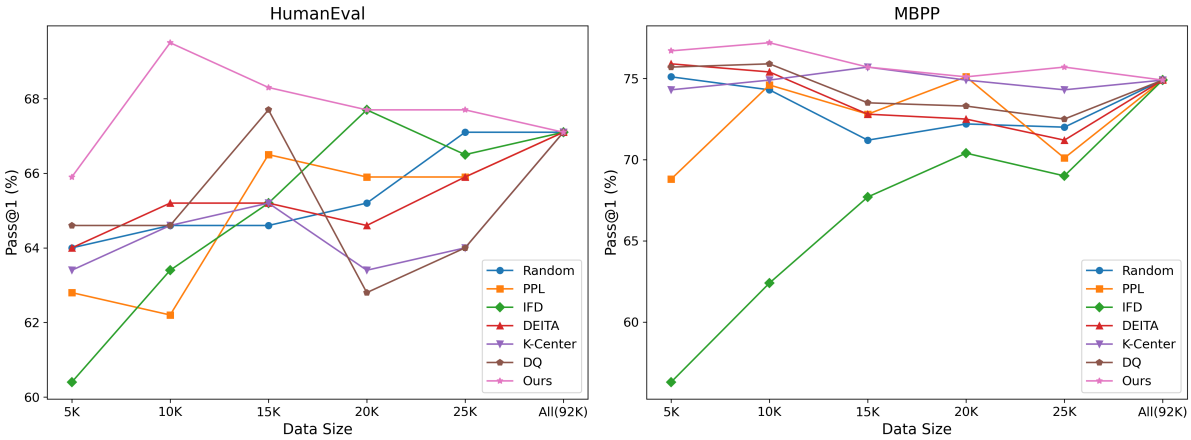

Figure 2: Performance comparison of different sampling methods with varying sampling quantities. Our method outperforms the other methods across different sample sizes, achieving the best performance with 10K samples.

putational overhead by optimizing the loss function in Equation 7 that ensures both distribution consistency with the original dataset and diversity within the selected subset.

5.3 Sampling Quantity

To investigate the impact of sampling quantity, we compared the experimental results with 5K, 10K, 15K, 20K, 25K, and the full 92K dataset, as shown in Figure 2. The experimental results reveal that the performance curves of all methods generally exhibit an initial increase followed by a decline, indicating the presence of low-quality data within the dataset. Moderate sampling (e.g., 10K data) effectively filters out low-quality samples, thus improving model performance, while excessive sampling (more than 25K data) may introduce noise, leading to performance degradation. When trained on the full dataset, the model achieved 67.1% on HumanEval and 74.9% on MBPP.

The results demonstrate that our method outperforms others across different sampling quantities, particularly peaking at 10K samples with 69.5% on HumanEval and 77.2% on MBPP. Compared to training on the full dataset, our method represents improvements of 2.4% and 2.3% respectively. These results validate the effectiveness of our data selection strategy, showing that introducing the parametric model into the data selection process can significantly enhance model performance.

As the sampling quantity increases from 5K to 20K, PPL and IFD show performance improvements on both HumanEval and MBPP benchmarks, indicating that "non-complex yet important" data samples also play a key role in improving performance. The improvement observed in these methods suggests that strategies focused solely on complex data sampling fail to yield the best performance, as they neglect simpler yet impactful samples. In contrast, our method considers both distribution consistency and diversity, ensuring that the sampled dataset not only includes complex data but also retains simpler yet essential samples, resulting in a more balanced performance gains.

The performance curves of DEITA, DQ and K-Center exhibit similar trends, with performance declining after exceeding 15K samples. Notably, K-Center maintains relatively good performance on the MBPP benchmark but fluctuates significantly on the HumanEval benchmark. This reflects the limitations of diversity-driven sampling strategies in programming semantic understanding. Purely diversity-based approaches neglect distribution consistency, potentially resulting in unstable performance on different benchmarks.

5.4 Cross-Architecture Generalization

To evaluate the robustness of our data selection strategy, we extend experiments to various code LLMs, including the CodeQwen1.5, Qwen2.5-Coder, and DeepSeek-Coder series, with scales ranging from 1.3B to 7B. We compare our method against the pre-trained base models (Base) and those fine-tuned on the full 92K dataset (All Data). Results are illustrated in Figure 3.

Our method demonstrates consistent effectiveness and superior generalization across diverse architectures. As shown in Figure 3, models fine-tuned with our method significantly outperform both the *Base* and *All Data* settings across all tested

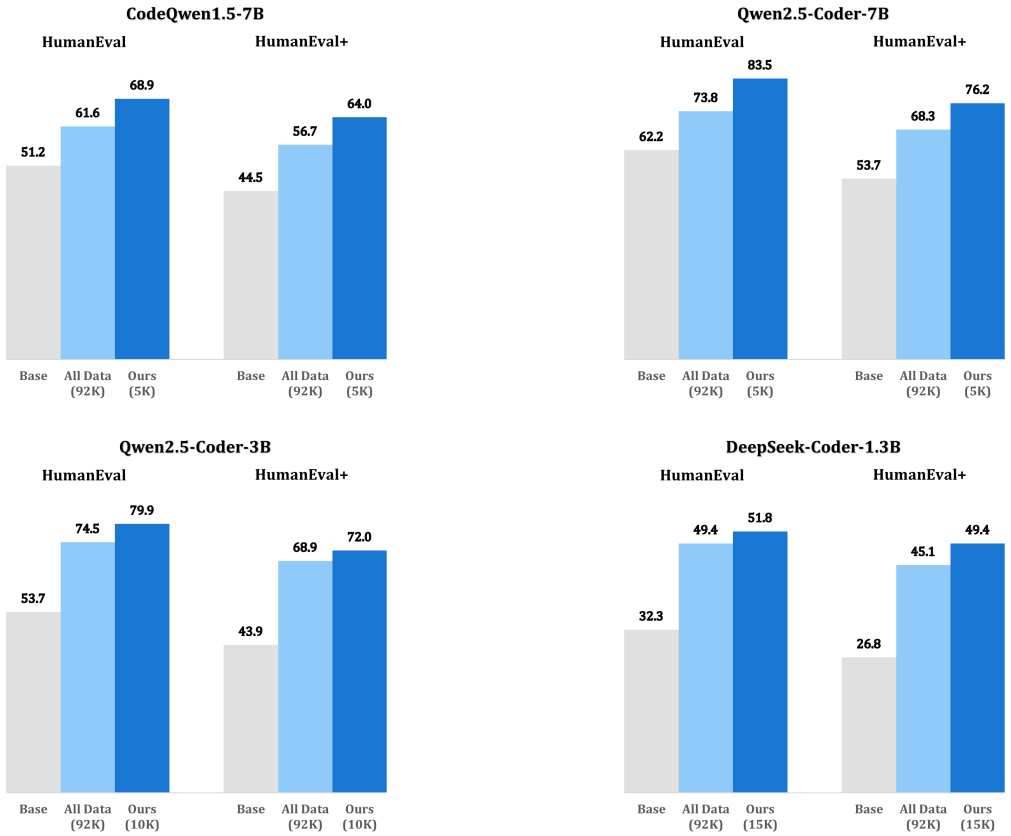

Figure 3: Generalization performance of our method across different model architectures and parameter scales.

scales. For instance, Qwen2.5-Coder-7B reaches 83.5% on HumanEval with only 5K selected samples, surpassing the *All Data* setting by a nearly 10 percentage point. Even for the 1.3B DeepSeek-Coder, our method maintains a clear advantage (49.4% vs. 45.1% on HumanEval+). This proves that our strategy effectively filters noise and redundancy, allowing models to focus on core programming patterns regardless of their underlying architecture or pre-training corpus.

We find a correlation between model capacity and data requirements for peak performance. While 7B models (e.g., Qwen2.5-Coder and CodeQwen1.5) achieve performance peak with just 5K samples, the smaller 1.3B model requires 15K to reach its optimal state. This stems from the fact that model capacity dictates the sensitivity to data volume and quality: leveraging strong generalization, 7B models can quickly distill core patterns from minimal data; conversely, 1.3B models require more samples to cover essential knowledge facets within their constrained parameter space. Crucially, both remain highly sensitive to data quality, as excessive low-quality samples can severely impair their performance.

In summary, these results confirm that our parametric feature-space selection is a model-agnostic, extensible solution. It significantly enhances code LLM performance at a fraction of the data cost, offering a practical tool for efficient fine-tuning.

6 Conclusion

In this work, we propose an efficient data selection strategy designed for code data. By optimizing parametric models, our approach ensures both distribution consistency between the selected subset and the original dataset, while simultaneously maximizing the diversity of the subset. Experimental results demonstrate that by using only 10K carefully selected data, our method achieves the best performance across all benchmarks, outperforming both other sampling methods and existing code LLMs. These findings underscore that the data quality is more important than the data quantity, and that an effective data selection strategies can significantly enhance both model performance and training efficiency. We hope that our study provides valuable insights for efficiently training code LLMs and contributes to advancing progress in related research fields.

Limitations

While our method shows significant advantages in code generation tasks, there are limitations that warrant further investigation. First, the current experimental validation is limited to Python-specific evaluation benchmarks. Due to expensive equipment rental costs, we have not yet conducted experiments on other programming languages (e.g., Java, C++) or larger-scale models (e.g., 34B parameters). More critically, our method does not directly validate the functional correctness of the code, potentially allowing flawed samples into the training set. In future work, we plan to integrate executable sandbox environments to rigorously verify code correctness and extend evaluations to a broader range of programming languages and model scales for a comprehensive assessment of the method's generalization capabilities.

Acknowledgments

We would like to express our sincere gratitude to those who have contributed to this work. Additionally, we utilize ChatGPT to review and refine sentence structures, which helped enhance the clarity and quality of the text.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zachary Ankner, Cody Blakeney, Kartik Sreenivasan, Max Marion, Matthew L Leavitt, and Mansheej Paul. 2024. Perplexed by perplexity: Perplexity-based data pruning with small reference models. *arXiv preprint arXiv:2405.20541*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and 1 others. 2024. Alpagasus: Training a better alpaca with fewer data. In *The Twelfth International Conference on Learning Representations*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, and 1 others. 2023. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. 2021. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*, pages 722–754. PMLR.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. 2024a. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14255–14273.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024b. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7595–7628.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023b. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36:21558–21572.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024b. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, and 1 others. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Keming Lu, Hongyi Yuan, Zheng Yuan, Runji Lin, Junyang Lin, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2024. # instag: Instruction tagging for analyzing supervised fine-tuning of large language models. In *The Twelfth International Conference on Learning Representations*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

Weijie Lv, Xuan Xia, and Sheng-Jun Huang. 2025. Data-efficient llm fine-tuning for code generation. *arXiv preprint arXiv:2504.12687*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Ozan Sener and Silvio Savarese. 2018. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, and 1 others. 2024. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Yifan Wang, Yafei Liu, Chufan Shi, Haoling Li, Chen Chen, Haonan Lu, and Yujiu Yang. 2024. Inscl: A data-efficient continual learning paradigm for fine-tuning large language models with instructions. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 663–677.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with oss-instruct. In *International Conference on Machine Learning*, pages 52632–52657. PMLR.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. Less: selecting influential data for targeted instruction tuning. In *Proceedings of the 41st International Conference on Machine Learning*, pages 54104–54132.

Yichen Xie, Han Lu, Junchi Yan, Xiaokang Yang, Masayoshi Tomizuka, and Wei Zhan. 2023. Active finetuning: Exploiting annotation budget in the pretraining-finetuning paradigm. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 23715–23724.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.

Yang Xu, Yongqiang Yao, Yufan Huang, Mengnan Qi, Maoquan Wang, Bin Gu, and Neel Sundaresan. 2023. Rethinking the instruction quality: Lift is what you need. *arXiv preprint arXiv:2312.11508*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Zhaojian Yu, Xin Zhang, Ning Shang, Yangyu Huang, Can Xu, Yishujie Zhao, Wenxiang Hu, and Qiufeng Yin. 2024. Wavecoder: Widespread and versatile enhancement for code large language models by instruction tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5140–5153.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, and 1 others. 2023a. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36:55006–55021.

Daquan Zhou, Kai Wang, Jianyang Gu, Xiangyu Peng, Dongze Lian, Yifan Zhang, Yang You, and Jiashi Feng. 2023b. Dataset quantization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17205–17216.

A Comparison with Competitive Code LLMs

We compare the performance of our method with several representative code LLMs, as summarized in Table 2. WizardCoder-CL was trained on the CodeLlama-Python base model using 78K instruction samples, while WaveCoder-DS and Magicoder-DS were both fine-tuned on DeepSeekCoder-Base with 20K and 75K samples, respectively. Despite these competitive models utilizing significantly larger training datasets, our approach achieves superior performance across all benchmarks using only 10K carefully selected samples.

These results emphasize that data quality remains a more critical factor than sheer quantity for instruction fine-tuning. By developing an effective data selection strategy, we can significantly enhance model capabilities while maintaining training efficiency. Our experiments confirm that employing a parametric model for data selection identifies high-quality training samples and substantially reduces computational resource consumption by filtering out redundant or low-quality data. Compared to models relying on massive training corpora, our method delivers better performance with a fraction of the data, demonstrating its effectiveness in resource-constrained scenarios.

B Detailed Ablation Studies

B.1 Impact of Different Feature Encoders

The choice of feature space is fundamental to parametric data selection. We compare our default approach (using *all-mpnet-base-v2* on instruction text I_i) against strategies using code-specific encoders (*jina-code-v2* and *CodeRankEmbed*) applied directly to code snippets C_i. The results are summarized in Table 3.

The data indicates that instruction-based encoding consistently outperforms code-based strategies. Notably, our method achieves 77.2% on MBPP, outperforming both code-based encoders by 4.7–6.8 percentage points. This confirms that for instruction fine-tuning, the feature space should align with the task intent (the instruction) rather than its implementation (the code).

We attribute this to two primary factors: First, instruction text directly defines the *intent space*, which aligns with human cognitive patterns and abstract logic. Selecting samples in this space ensures better coverage of task types and core logic, directly aligning training objectives with evaluation objectives (i.e., generating code from instructions). Second, code snippets often contain significant *implementation noise*, such as specific syntax choices, library calls, or naming conventions that are irrelevant to the core task logic. Encoding implementation-level details can mislead the selection process, leading to a subset that is homogeneous in coding style but lacks semantic diversity, thereby weakening the model's instruction-following capability.

B.2 Sensitivity Analysis of Temperature τ

The temperature parameter τ regulates the sharpness of the similarity distribution in the loss function, significantly impacting the optimization dynamics of the parametric model. We evaluate $\tau \in \{0.04, 0.07, 0.2, 0.5\}$, with results summarized in Table 4.

Performance follows a non-monotonic trend, peaking at $\tau = 0.07$ across all benchmarks. This setting represents an ideal balance between distribution sharpness and optimization stability. Specif-

Model	Params	Data Size	HumanEval	HumanEval+	MBPP	MBPP+
CodeLlama-Python	7B	-	37.8%	35.4%	59.8%	46.8%
WizardCoder-CL	7B	78K	50.6%	45.1%	58.5%	49.5%
DeepseekCoder-Base	6.7B	-	47.6%	39.6%	72.0%	58.7%
WaveCoder-DS	6.7B	20K	61.0%	54.9%	75.9%	60.9%
Magicoder-DS	6.7B	75K	66.5%	60.4%	75.4%	61.9%
Ours-DS	6.7B	10K	**69.5%**	**63.4%**	**77.2%**	**63.2%**

Table 2: Comparison of different code LLMs. The results for Magicoder-DS are taken from the original paper (Wei et al., 2024). We re-evaluated WaveCoder-DS as the results in their original paper (Yu et al., 2024) were incomplete. Results for other models are sourced from the EvalPlus Leaderboard (Liu et al., 2023).

Feature Encoder	Encode Type	HumanEval	HumanEval+	MBPP	MBPP+
all-mpnet-base-v2	Instruction	**69.5**	**63.4**	**77.2**	**63.2**
jina-code-embeddings-v2	Code	67.1	62.2	70.4	54.8
CodeRankEmbed	Code	64.6	58.5	72.5	59.3

Table 3: Impact of different feature encoders on model performance.

Temperature τ	HumanEval	HumanEval+	MBPP	MBPP+
0.04	67.1	61.0	74.3	60.3
0.07	**69.5**	**63.4**	**77.2**	**63.2**
0.2	67.5	63.2	74.5	60.9
0.5	68.3	62.8	74.9	60.1

Table 4: Ablation study results for the temperature parameter τ.

ically, an excessively low τ (0.04) amplifies minor similarity differences, creating an overly sharp distribution that forces the optimization to focus solely on the immediate nearest neighbors while ignoring the broader neighborhood structure. This often leads to local minima and numerical instability.

Conversely, a high τ (0.2 or 0.5) flattens the distribution, narrowing the contrast between relevant and irrelevant samples. This weakens both the attraction toward data clusters and the repulsion between parameters, resulting in blurred gradient signals that fail to guide the precise positioning of anchors for optimal representativeness. Thus, $\tau = 0.07$ is adopted as the optimal setting to provide the most discriminative and stable gradient signals.