Code Less, Align More: Efficient LLM Fine-tuning for Code Generation with Data Pruning

Anonymous ACL submission

Abstract

Recent work targeting large language models (LLMs) for code generation demonstrated that increasing the amount of training data through synthetic code generation often leads to exceptional performance. In this paper we explore data pruning methods aimed at enhancing the efficiency of model training specifically for code LLMs. We present techniques that integrate various clustering and pruning metrics to selectively reduce training data without compromising the accuracy and functionality of the generated code. We observe significant redundancies in synthetic training data generation, where our experiments demonstrate that benchmark performance can be largely preserved by training on only 10% of the data. Moreover, we observe consistent improvements in benchmark results through moderate pruning of the training data. Our experiments show that these pruning strategies not only reduce the computational resources needed but also enhance the overall quality code generation.

1 Introduction

The performance of large language models (LLMs) is heavily dependent on the size and quality of their training datasets, as highlighted by recent studies on scaling laws (Achiam et al., 2023; Zhang et al., 2024). State-of-the-art code LLMs, such as CodeAlpaca (Chaudhary, 2023), WizardCoder (Luo et al., 2024), and MagicCoder (Wei et al., 2023), have achieved remarkable performance by significantly expanding their supervised fine-tuning datasets through synthetic code generation. Various synthetic code generation approaches have been developed, including the Self-Instruct technique (Wang et al., 2022), Evol-Instruct (Xu et al., 2023a), and OSS-Instruct (Wei et al., 2023). However, such scaling approaches not only increase the training cost but also demands substantial computational resources, making it expensive and less accessible.

Achieving optimal performance in fine-tuned models for downstream tasks often relies on large, high-quality datasets. Recently, there has been a growing interest in more efficient fine-tuning methods for large language models (LLMs). One recent work introduces the Superficial Alignment Hypothesis (Zhou et al., 2023), which suggests that most knowledge in LLMs is acquired during pretraining, and only minimal instruction tuning data is required to align models with human preferences. Promising strategies to reduce computational demands include parameter-efficient fine-tuning (PEFT) methods, which reduce the number of parameters needed for training (Fu et al., 2023; Hu et al., 2021). Another research direction uses active learning to iteratively select data samples during training, thereby enhancing model learning (Su et al., 2022; Diao et al., 2023). These methods primarily aim to improve model accuracy through iterative processes, requiring multiple rounds of training and data selection.

Data selection and pruning methods have also been well-explored in literature, with evidence suggesting that careful pruning can sometimes even surpass the performance of using the full dataset (Penedo et al., 2024; Wang et al., 2023). Moreover, many of these methods are computationally intensive such as supervised metrics that involves multiple times of model training to keep track of loss and gradients (Xia et al., 2024; Pruthi et al., 2020) or heavy sampling method with Monte Carlo (Schoch et al., 2023), limiting their scalability. Practical pruning methods that aims for large-scale data have been investigated in the contexts of LLM pretraining (Das and Khetan, 2023; Penedo et al., 2024) and fine-tuning (Chen et al., 2024; Schoch et al., 2023) datasets, image datasets (Moser et al., 2024; Meding et al., 2021), and vision-text training datasets (Wang et al., 2023), and demonstrate success by applying clustering and by choosing proper indicator functions.

Despite these advances, there remains a gap in efficient pruning strategies specifically tailored for coding datasets. Most large-scale code datasets are synthetically generated, resulting in many data samples with similar lexical appearances due to consistent formatting and style. Large-scale synthetic datasets commonly used for training code LLMs often suffer from significant redundancy and noise (Wang et al., 2023). This redundancy arises from the impracticality of verifying the functional correctness of each program, leading to a substantial portion of instruction-code pairs being noisy. Therefore, enhancing data efficiency through careful selection and pruning of data samples is crucial for improving model performance without relying on excessively large datasets.

In this work, we present a scalable and effective data pruning method to enhance code generation in large language models. Our approach clusters data samples based on problem instructions and their code solutions, applying dimensionality reduction to reduce computational load. We then select a representative subset from each cluster using various pruning metrics. Experiments on large-scale datasets and evaluations on downstream coding tasks show that our method maintains or even improves model performance while significantly reducing training data. Our contributions and key findings are summarized as follows:

- We are the first to study data pruning for large-scale synthetic code fine-tuning. We create an efficient and scalable pruning strategy based on unsupervised learning methods.
- We find large redundancies in synthetic generated code datasets, as training on just 10% retains most benchmark performance, with slight degradation of 3.9% on HumanEval and 1.5% on MBPP compared with using all data.
- We observe consistent improvement by moderately pruning the dataset, leading to improvements of up to 2.7% on HumanEval and 3.5% on MBPP compared with using all data.
- We perform detailed ablation studies, where results demonstrate the clustering algorithm to be critical, while pruning metrics to be less important.

2 Related Work

In this section, we review the advancements of large language models (LLMs) for code generation in Section 2.1 and review prior work on instructional finetuning in Section 2.2. Finally, we discuss earlier research on data selection and pruning methods in Section 2.3.

2.1 Large Language Models for Code Generation

Great advancements have been achieved in improving Large Language Models (LLMs) for code generation. Codealpaca (Chaudhary, 2023) extends the capabilities of the LLaMA model (Touvron et al., 2023a) by incorporating 20,000 instruction-following data points generated through the Self-Instruct technique (Wang et al., 2022), which aligns language models with self-generated instructions. CodeLlama (Roziere et al., 2023) further enhances this methodology by fine-tuning from LLaMA2 (Touvron et al., 2023b), utilizing 14,000 instruction-following data points also generated via the Self-Instruct technique.

Wizardcoder (Luo et al., 2024) utilizes the Evol-Instruct method (Xu et al., 2023a) to evolve the Codealpaca dataset further. This technique iteratively evolves instruction-following data in both depth and breadth dimensions. On the other hand, Magicoder (Wei et al., 2023) employs the OSS-Instruct technique to create instruction-following data from unlabeled open-source code snippets, constructing a dataset of 75,000 samples based on the StarCoder dataset (Lozhkov et al., 2024).

2.2 Instructional Fine-tuning

Fine-tuning language models with instructional datasets has emerged as a powerful technique, offering notable improvements in model performance and alignment with human preferences and safety. By exploring a diverse array of instructional tasks, (Wei et al., 2021) demonstrated a significant enhancement in zero-shot performance on unseen tasks through fine-tuning. Building on this, (Chung et al., 2024) showed that scaling both the number of tasks and the model size can lead to substantial performance gains across different model architectures. (Peng et al., 2023) further advanced this field by leveraging large language models (LLMs) to generate high-quality instruction-following data, resulting in improved zero-shot performance on new tasks.

A recent study (Zhou et al., 2023) introduces the Superficial Alignment Hypothesis, which posits that the bulk of knowledge in LLMs is acquired during pretraining. It further suggests that minimal fine-tuning data is sufficient to align these models with human preferences. The study demonstrates a noteworthy enhancement in LLM performance with just 1,000 high-quality instruction data points. Subsequently, a plethora of research endeavors have concentrated on refining dataset quality through diverse filtering methodologies for general instruction following (Xu et al., 2023b; Chen et al., 2024; Liu et al., 2023b).

2.3 Data Pruning for Efficient Training

Various pruning methods have been explored for selecting more informative samples for model training, each tailored to different scenarios. Data clustering has been widely used as a highly effective technique for data pruning. TLDR (Wang et al., 2023) utilized KMeans clustering to group similar data points and uniformly sampled from each cluster. They employ Image-Text Matching (ITM) scores to identify suitable vision-text pairs, offering another perspective on sample selection. DEFT (Das and Khetan, 2023) utilizes unsupervised core-set selection for clustering-based data-efficient fine-tuning of LLMs. This approach significantly enhances data efficiency in fine-tuning for text-editing applications.

Metrics like Hardness (Sorscher et al., 2022), Instruction Following Difficulty (IFD) (Li et al., 2023) (Li et al., 2023), and SuperFiltering (Li et al., 2024) focus on identifying "hard" samples that are either difficult to learn or easy to forget, tracking each data sample throughout training. In addition to these, sample influence metrics such as LESS (Xia et al., 2024) and TracIn (Pruthi et al., 2020) monitor model gradients and the impact of individual samples, albeit with significant computational overhead for large models and datasets. Quality metrics from external oracles (Chen et al., 2024; Liu et al., 2023b), leverage strong language models like ChatGPT for data selection. However, utilizing external oracles may not always be feasible due to cost constraints.

3 Methodology

Our goal is to select high-quality, representative data samples so that training on these subsets yields performance that is comparable to or better than training on the entire dataset. The overview of efficient data pruning for fine-tuning LLMs with large scale datasets is illustrate in Figure 1. First, we use an embedding model to project the instruction-code pairs into a vector representation. We further reduce the dimension of feature representation to reduce computation complexity of the following steps. We then apply clustering to identify and group up similar data samples. Finally, we applied pruning metrics to further reduce data size. The detail pseudo code is in Algorithm 1.

When dealing with coding datasets, two primary selection directions can be considered: syntactical and semantic. Selecting programs that are syntactically different but semantically equivalent, or vice versa, can be inefficient. Our design will focus on identifying syntactical differences. Detecting semantic differences between programs typically requires fuzzing techniques (Chen et al., 2018), which involve creating larger test samples and executing programs to group them based on behavior. This approach contradicts our objective of reducing computational costs. Therefore, our method emphasizes syntactical analysis to achieve efficient and effective data selection.

Algorithm 1 Data Pruning Algorithm

1: Initialize $Embbedding$, Compression $Ratio$
2: Initialize $selected \leftarrow []$
3: $X \leftarrow$ PCA($Embedding$)
4: $Cluster \leftarrow$ ClusterAlgo(X)
5: **for** each $idx, items$ in $Cluster$ **do**
6:     $score \leftarrow$ PruningMetrics($item$)
7:     $remain \leftarrow$ Random($items$, prob=$score$)
8:     Update $Cluster[ids] \leftarrow remain$
9:     Append $selected \leftarrow remain$
10: **end for**
11: **Output:** $selected$

3.1 Dimension Reduction

We convert each instruction-code pair into vector representation using a embedding model from raw text to enhance the efficiency of clustering and computation of pruning metrics (Naik, 2024). Recent research indicates that distances based on LLM embeddings effectively capture syntactic differences. To address the computational complexity, we employ Principle Component Analysis (PCA) (Maćkiewicz and Ratajczak, 1993) to reduce the dimensionality of the vector representations, as representations extracted from LLMs often exceed a thousand dimensions. Moreover, this approach prevents the subsequent utilization of several pruning metrics, which involve kernel methods, from being

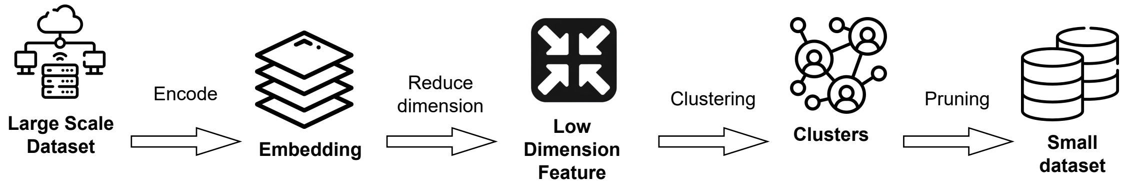

Figure 1: The overview of efficient data pruning for fine-tuning LLMs with large scale datasets. First, We reduce the encode instruction-following data into embedding and reduce the dimension of feature representation. Second, we apply clustering to identify and group up similar data samples. Finally, we applied pruning metrics to further reduce data size.

hindered in high-dimensional spaces by the curse of dimensionality.

3.2 Clustering

Clustering is a critical step in our methodology to group similar instruction-code pairs, which facilitates the selection of diverse and representative samples. Before clustering, we normalize the vector representations to ensure that each feature contributes equally to the distance calculations. From each cluster, we then sample instruction-code pairs to create a subset that is representative of the entire dataset. The sampling strategy is further decided by different pruning metrics.

3.2.1 KMeans

The KMeans algorithm (Kanungo et al., 2002) partitions data into k clusters. By minimizing the within-cluster sum-of-squares, KMeans ensures that each cluster is as compact as possible. The main advantage of KMeans is its scalability and efficiency in handling large datasets.

3.2.2 Agglomerative Clustering

Agglomerative Clustering (Müllner, 2011) builds nested clusters with linkage criteria. This method is advantageous since it does not require the number of clusters to be specified a priori. This flexibility allows for a more nuanced selection of representative samples, which is beneficial for maintaining the quality of the dataset.

3.2.3 HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) (Rahman et al., 2016) performs clustering based on the concept of core samples, which are samples located in high-density areas measured by a distance metric. This approach aligns well with our design hypothesis to find the most syntactically representative data samples. Notably, HDBSCAN removes noisy samples not clustered into core samples as outliers.

3.3 Pruning Metrics

The criteria of choosing pruning metrics continually aligns with the idea of detecting syntactic difference and find most representative samples. We explain the pruning metrics explored in our experiments in the following sections.

3.3.1 Diversity Metric

We use a distance-based metric that simply evaluates the diversity score of a single instance shown as follow,

$$d_i = \min_{\mathbf{x} \in \mathcal{K} \setminus \{\mathbf{x}_i\}} \text{dist}(\mathbf{x}_i, \mathbf{x}), \quad (1)$$

where x_i is the vector representation, *dist* is a distance function, K represents selected query set within the dataset cluster, and d_i is the diversity score of a sample x_i. We use the dot product of the embeddings as the distance function as our embeddings are normalized prior to pruning.

3.3.2 Density Metric

We applied kernel density estimation (KDE) to measure the density of samples in the feature space. KDE estimates the probability density function of a random variable. The density score for a sample $\mathbf{x}_i$ is given by,

$$\rho(\mathbf{x}_i) = \frac{1}{nh^d} \sum_{j=1}^{n} K\left(\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right), \quad (2)$$

where K is the kernel function, h is the bandwidth parameter, d is the dimension of the feature space, and n is the total number of samples. The kernel function K (typically a Gaussian) measures the influence of nearby points on the density estimate. A high density score indicates that a sample is located in a region with many similar instances, suggesting it is less critical for maintaining diversity.

Model	Training	Benchmark		Improvement Over Base	
	Tokens	HumanEval (+)	MBPP (+)	HumanEval (+)	MBPP (+)
GPT-3.5 Turbo	-	72.6 (65.9)	81.7 (69.4)	-	-
GPT-4 Turbo	-	**85.4** (**81.7**)	**83.0** (**70.7**)	-	-
DeepSeek-Coder-Base	-	47.6 (39.6)	70.2 (56.6)	-	-
DeepSeek-Coder-Instruct	2B	73.8 (70.1)	72.7 (63.4)	26.2 (30.5)	2.5 (6.8)
Magicoder-DS	90M	66.5 (60.4)	75.4 (61.9)	18.9 (20.8)	5.2 (5.3)
Magicoder$\mathcal{S}$-DS	240M	**76.8** (**70.7**)	**75.7** (**64.4**)	**29.2** (**31.1**)	**5.5** (**7.8**)
Ours (full data)	234M	74.3 (70.8)	74.5 (62.3)	26.7 (31.2)	4.3 (5.7)
Ours (90%)	192M	**77.0** (**71.6**)	76.9 (**64.0**)	**29.4** (**32.0**)	6.7 (**7.4**)
Ours (50%)	106M	71.0 (64.0)	**78.0** (**64.0**)	23.4 (24.4)	**7.8** (**7.4**)
Ours (10%)	21M	70.4 (65.0)	73.0 (60.2)	22.8 (25.4)	2.8 (3.6)
Ours (1%)	2M	64.6 (58.0)	74.3 (61.9)	17.0 (18.4)	4.1 (5.3)

Table 1: $pass@1$ (%) results of different LLMs on HumanEval (+) and MBPP (+) with greedy decoding. We directly use results from prior work (Guo et al., 2024; Wei et al., 2023). All our results are reported using the HDBSCAN clustering algorithm with the diversity pruning metric (HDBSCAN-diversity). To account for the randomness of clustering and training, we report the averaged results from three runs evaluated with EvalPlus (Liu et al., 2023a).

3.3.3 Random

The simplest baseline is random selection, where we randomly sample data from the selected cluster or entire training dataset (without clustering) for instruction tuning.

4 Experiments

In this section, we first present the experimental setup in Section 4.1, followed by our primary findings in Section 4.5. Here, we highlight the performance improvements of our pruning methods compared to full dataset training across four datasets: MBPP(+), and HumanEval(+). We also compare the $pass@1$ scores with baseline methods at various compression ratios.

4.1 Setup

We employed DeepSeek-Coder-Base 6.7B (Guo et al., 2024) as the base model due to its superior performance among open-source models. We used PCA (Maćkiewicz and Ratajczak, 1993) algorithm in all experiments and reduce the dimension to 10. To account for randomness in clustering algorithm and training, we repeat each experiment 3 times and report the average and standard deviation.

4.2 Training

Datasets In our experiment, we adopt two synthetic code dataset as training data: Magicoder-OSS-Instruct-75K [1] (MIT License) and Magicoder-Evol-Instruct-110K [2] (Apache-2.0 License). Together we have a combined 185k entries in total as our target large scale dataset.

We fine-tune the base model by combining and shuffling the two training dataset. This is different as in the original Magicoder (Wei et al., 2023) implementation, where they first fine-tune the base models for 2 epochs on OSS-Instruct data and continue training for 2 more epochs on Evol-Instruct data. We note that despite such difference in our implementation details, our full dataset performance closely matches the Magicoder$\mathcal{S}$-DS results.

Training Training is conducted with 16 NVIDIA A100-80GB GPUs through the Distributed Data Parallel (DDP) module from PyTorch. We set the learning rate at 5e-5 with 15 warmup steps and a linear learning rate scheduler. We use Adam (Kingma and Ba, 2014) as our optimizer with full parameter updates and truncate sequence length longer than 4096 tokens. We use a batch size of 512 samples (Wei et al., 2023) when the dataset size exceeds $\geq 10\%$ of the original size, and a batch size of 32 (Zhou et al., 2023) for heavily pruned small-scaled data experiments in Figure 3. We fine-tune for 2 epochs regardless of the dataset size.

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

[2] https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K

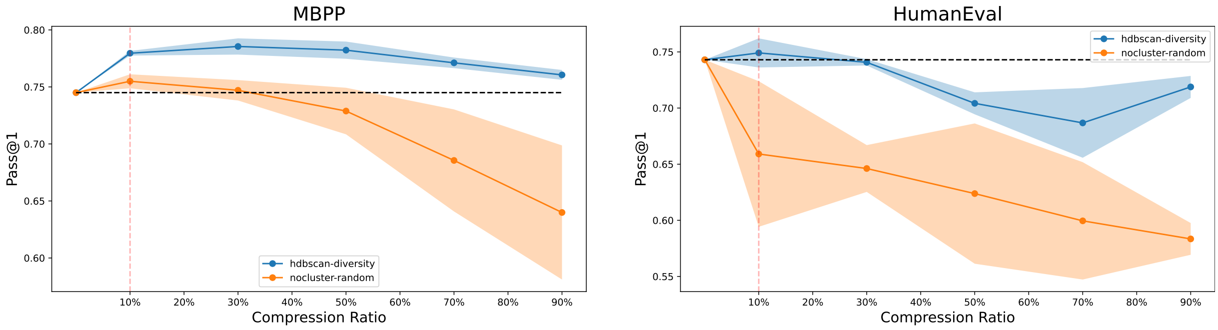

Figure 2: Performance comparison of HDBSCAN-diversity and nocluster-random methods across different benchmarks. Our strategy outperform the baseline across different datasets with a large margin. We also maintain better or equivalent performance compare to full dataset even at the size of 10% on MBPP. The $pass@1$ metric is plotted against varying compression ratios, demonstrating the robustness and effectiveness. HumanEval presents larger variance across experiments possibly due to less problems entries.

4.3 Evaluation

Datasets HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) are two of the most widely used benchmarks for code generation. The two datasets contains 164 and 1401 problems respectively. Each task in these benchmarks includes a task description (e.g., docstring) as the prompt, where LLMs generate corresponding code whose correctness is checked by a handful of test cases. Because tests in these benchmarks can be insufficient, for more rigorous evaluation, we use HumanEval+ and MBPP+, both powered by EvalPlus (Liu et al., 2023a) to obtain 80× and 35× more tests, respectively.

Metric Following prior work (Chen et al., 2021; Liu et al., 2023a), for each experiment we use the unbiased pass@k estimator shown as follow and mainly focus on comparing $pass@1$ metric:

$$pass@k := \mathbb{E}_{\text{Problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]. \tag{3}$$

Inference We employ the EvalPlus (Liu et al., 2023a) inference script with sanitation postprocessing. We adopted the vLLM (Kwon et al., 2023) framework and use greedy decoding for every code generation. The inference engine is setup with bf16 dtype, tensor parallel size of 2 and a maximum length of 4096.

4.4 Implementation Details

In our experiment, the PCA reduction is fitted on the benchmark dataset and then apply the projection to the instruction data. We used the OpenAI *text-embedding-ada-002* embedding model to encode data. All the clustering and kernel density estimation parameters are as default in sklearn (Pedregosa et al., 2011). For algorithms that requires choosing an optimal number of clusters (such as KMeans) is crucial, we utilize the Elbow method (Roy, 1953) to find the point where adding more clusters does not significantly improve the variance explained. For pruning metrics, we applied the Scott's Rule (Scott, 2010), a normal-reference rule for deciding the Gaussian kernel bandwidth, for kernel density estimation and random select 10% of the dataset as query set (K) for diversity metric.

4.5 Main Results

Table 1 presents the $pass@1$ results of different leading code LLMs on the HumanEval and MBPP benchmarks, computed with greedy decoding. All our results are reported using the HDBSCAN clustering algorithm with the diversity pruning metric (HDBSCAN-diversity). To account for the randomness of clustering and training, we report the averaged results from three runs. Notably, slight pruning of the training data could yield a performance improvement of up to 2.7% on HumanEval and 3.5% on MBPP compared to training with the full dataset. We further show that benchmark accuracy can be largely retained with 10% of the dataset, with slight degradation of 3.9% on HumanEval and 1.5% on MBPP compared with using the full training data. Even with just 1% of the data ($\sim$ 700 samples), our method maintains competitive performance and achieves large improvements over the base model, underscoring the efficiency of our pruning strategy.

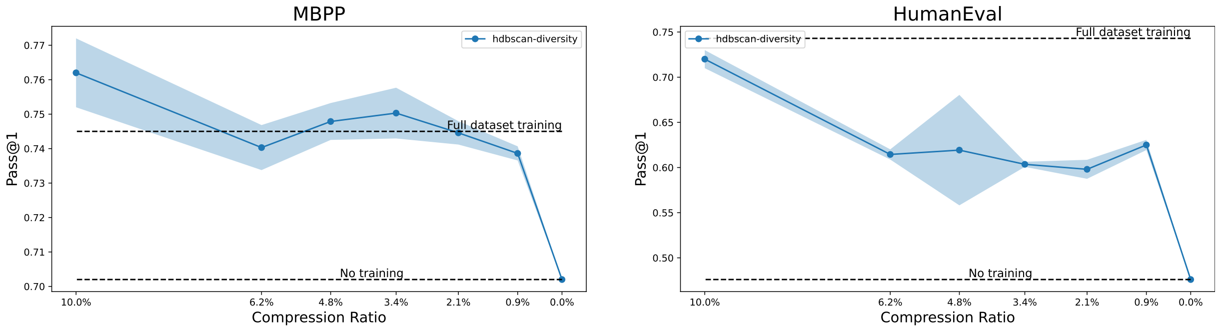

Figure 3: Comparison of performance under extreme data pruning conditions on the MBPP and HumanEval benchmarks. The $pass@1$ score on MBPP shows that even with just 1% of the data, our method achieves nearly equivalent performance to the full dataset, with a 4.1% improvement over the base model. On the HumanEval benchmark, while the performance with 1% of the data degrades compared to the full dataset training, it still achieves an 17.0% improvement over the base model.

Figure 2 illustrates the detail of our pruning methods across four datasets: MBPP, MBPP+, HumanEval, and HumanEval+. Each subplot compares the $pass@1$ scores of the HDBSCAN-diversity method with the nocluster-random baseline at various compression ratios. HDBSCAN-diversity method consistently outperforms the nocluster-random baseline. The performance typically improves with slight compression, peaking around 10-20%, and then gradually declines. This trend highlights the robustness of the HDBSCAN-diversity method, maintaining higher $pass@1$ scores than full dataset even at 90% compression.

We further examine how our data pruning method performs when pushed to the extreme, aiming to achieve the smallest possible dataset size on the MBPP benchmark. The results are presented in Figure 3. Remarkably, we found that even with just 1% of the data, our method achieves a 4.1% improvement over the base model, which is nearly equivalent to training on the full dataset. This demonstrates the robustness of our pruning method, highlighting its ability to maintain high performance with minimal data, thus significantly reducing the computational resources required.

Overall, these results demonstrate the effectiveness of data pruning strategy in preserving critical data features and maintaining model performance under significant data reduction, making it a superior choice for coding dataset pruning.

5 Ablation Studies

Our research includes four ablation studies designed to evaluate the impact of (1) clustering algorithms (2) pruning metrics (3) dimension reduction (4) input for vector representation on the effectiveness of data pruning. In the studies, we will mainly focus on the MBPP benchmark since it provides more stable and consistent results.

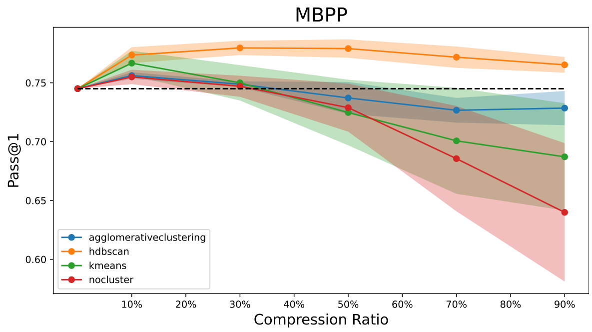

Figure 4: $pass@1$ on the MBPP benchmark comparing across different clustering algorithms and varied compression ratios of the training dataset. HDBSCAN demonstrate strong robustness in maintaining higher $pass@1$ scores compared to full dataset at the compression ratio of 90%.

5.1 Compare Clustering Algorithm

In Figure 4, we present the results of applying different clustering algorithms without additional pruning metrics. The algorithms evaluated include Agglomerative Clustering, HDBSCAN, KMeans, and a baseline with no clustering (nocluster).

The results demonstrate that clustering algorithms generally improve performance compared to the nocluster baseline, particularly at higher compression ratios. HDBSCAN consistently maintains higher $pass@1$ scores, showcasing its robustness in preserving critical data features. KMeans and Agglomerative Clustering also perform well, though with higher variability. These findings highlight the importance of clustering algorithms in enhancing data efficiency for coding datasets.

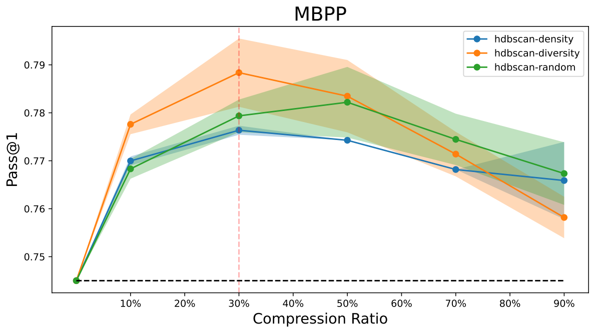

Figure 5: Comparison of different pruning metrics using HDBSCAN clustering algorithms. Diversity metric has marginal advantage but its benefit may be limited and dependent on the clustering algorithm.

5.2 Compare Pruning Metrics

We examine the impact of different pruning metrics on model performance. Using HDBSCAN clustering algorithm, we assess how these metrics influence performance as the data size decreases, as illustrated in Figure 5. The results indicate that the effectiveness of pruning metrics varies across different compression ratio. While Diversity metrics show slight improvements over other metrics, the margin of improvement is not substantial and only works between 10-40% compression ratio. This suggests that while more sophisticated pruning metrics can offer some benefits, their impact may be limited and also dependent on the clustering algorithm used.

5.3 Effect of PCA

In Table 2, we evaluate the impact of applying Principal Component Analysis (PCA) on the performance of the KMeans clustering algorithm and Density metric at the compression ratio of 50%. The findings indicate that applying PCA generally degrades performance in terms of $pass@1$ scores for less than 0.6% on MBPP, and moderate negative impact of 4.3% on HumanEval. We hypothesize that the observed impact might be due to the imbalance between the MBPP and HumanEval datasets used for PCA training. Since the HumanEval dataset is significantly smaller than the MBPP dataset, it results in suboptimal extraction of principal components for HumanEval-like data.

Nonetheless, reducing the dimension from 1536 to 10 leads to $\sim$12x speed up for KMeans. HDBSCAN clustering without PCA does not complete within 4 hours, thus we do not report its numbers.

	No PCA	PCA
Dimension	1536	10
Runtime	1307 sec	183 sec
MBPP (+)	74.4 (63.3)	73.8 (62.4)
HumanEval (+)	71.8 (65.0)	67.5 (62.5)

Table 2: Comparison of $pass@1$ scores, dimension, and data pruning runtime (excludes embedding and training) at 50% compression ratio for KMeans clustering with and without PCA (averaged over 3 runs).

5.4 Embeddings for Instruction or Code

In Table 3, we investigate the influence of various inputs on the embedding model. Specifically, we examine the effects of using only the instruction, only the code solution, or both as inputs for generating embeddings. Our findings indicate that combining both instructions and code as embedding inputs yields better performance compared to using either one alone. There are no significant differences in the results when using only instructions or only code. This suggests that even though instructions and code samples often correspond closely, it is crucial to maintain diversity and select informative samples from both during data pruning.

Feature Type	MBPP (+)	HumanEval (+)
Both	76.3 (62.5)	73.1 (69.6)
Instruction	74.0 (63.7)	69.1 (63.6)
Code	74.1 (62.7)	69.2 (63.3)

Table 3: $pass@1$ scores for different embedding inputs with 50% compression ratio using KMeans clustering. Using both instruction and code brings slight benefits.

6 Conclusion

This study presents an efficient data pruning strategy designed to improve the efficiency of fine-tuning large language models on coding datasets. Our results demonstrate that advanced clustering and pruning techniques can significantly improve data efficiency in LLMs, reducing computational costs while maintaining performance. Future work could focus on enhancing data quality by generating more informative data from clusters with low pruning metrics. We hope our findings provide valuable insights for developing more effective and scalable strategies in training code-focused LLMs, further enhancing synthetic data generation and the efficiency of human annotations.

Limitations

One of the key limitations of our study is the inherent randomness from the clustering algorithms and training framework. Due to computational constraints, we only performed three runs and averaged the results for each of our experiments. While this approach provides a general indication of performance, it may not fully capture the variability and could lead to less accurate conclusions. More extensive experimentation with a larger number of runs would be necessary to achieve a higher degree of confidence in the results.

Throughout our experiments, we closely follow the hyperparameters described in (Wei et al., 2023), using a batch size of 512 samples and training for 2 epochs. However, such a high batch size results in only a few gradient updates when training on smaller datasets. Therefore, we switch to a lower batch size of 32, as recommended in (Zhou et al., 2023), when our pruned dataset is less than 10% of the original size. We acknowledge that different hyperparameter settings could affect training outcomes and defer the determination of optimal hyperparameter settings for various training data sizes as future work.

Potential Risks

This study focus exclusively on English prompts for Python code generation, thus prompts in other languages might not produce accurate or functional code. Additionally, the lack of safety alignment means there is a risk of generating malicious code or harmful language, which could lead to security vulnerabilities or unintended consequences. Code generation using LLMs carries inherent risks, such as producing incorrect or suboptimal code, failing to adhere to best practices, or introducing security flaws. Furthermore, LLMs may inadvertently propagate biases present in their training data, leading to biased outcomes in the generated code.

Use of AI Assistants

ChatGPT was utilized to refine paper writing and generate code templates for drawing figures. The authors took careful attention to ensure that AI-generated contents are accurate and align with the authors intentions.

References

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 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.

Chen Chen, Baojiang Cui, Jinxin Ma, Runpu Wu, Jianchao Guo, and Wenqian Liu. 2018. A systematic review of fuzzing techniques. *Computers & Security*, 75:118–137.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. 2024. Alpagasus: Training a better alpaca with fewer data. *Preprint*, arXiv:2307.08701.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 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, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Devleena Das and Vivek Khetan. 2023. Deft: Data efficient fine-tuning for large language models via unsupervised core-set selection. *arXiv preprint arXiv:2310.16776*.

Shizhe Diao, Pengcheng Wang, Yong Lin, and Tong Zhang. 2023. Active prompting with chain-of-thought for large language models. *arXiv preprint arXiv:2302.12246*.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12799–12807.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. 2002. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. 2024. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. *arXiv preprint arXiv:2402.00530*.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2023. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. *arXiv preprint arXiv:2308.12032*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023a. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2023b. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*.

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

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

Andrzej Maćkiewicz and Waldemar Ratajczak. 1993. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342.

Kristof Meding, Luca M Schulze Buschoff, Robert Geirhos, and Felix A Wichmann. 2021. Trivial or impossible–dichotomous data difficulty masks model differences (on imagenet and beyond). *arXiv preprint arXiv:2110.05922*.

Brian B Moser, Federico Raue, and Andreas Dengel. 2024. A study in dataset pruning for image super-resolution. *arXiv preprint arXiv:2403.17083*.

Daniel Müllner. 2011. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*.

Atharva Naik. 2024. On the limitations of embedding based methods for measuring functional correctness for code generation. *arXiv preprint arXiv:2405.01580*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Guilherme Penedo, Hynek Kydlíček, Leandro von Werra, and Thomas Wolf. 2024. Fineweb.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930.

Md Farhadur Rahman, Weimo Liu, Saad Bin Suhaim, Saravanan Thirumuruganathan, Nan Zhang, and Gautam Das. 2016. Hdbscan: Density based clustering over location based services. *arXiv preprint arXiv:1602.03730*.

Samarendra Nath Roy. 1953. On a heuristic method of test construction and its use in multivariate analysis. *The Annals of Mathematical Statistics*, 24(2):220–238.

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*.

Stephanie Schoch, Ritwick Mishra, and Yangfeng Ji. 2023. Data selection for fine-tuning large language models using transferred shapley values. *arXiv preprint arXiv:2306.10165*.

David W Scott. 2010. Scott's rule. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):497–502.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari Morcos. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35:19523–19536.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv:2209.01975*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Alex Jinpeng Wang, Kevin Qinghong Lin, David Junhao Zhang, Stan Weixian Lei, and Mike Zheng Shou. 2023. Too large; data reduction for vision-language pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3147–3157.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

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. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023a. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

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

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. 2024. When scaling meets llm finetuning: The effect of data, model and finetuning method. *arXiv preprint arXiv:2402.17193*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023. Lima: Less is more for alignment. *Preprint*, arXiv:2305.11206.

A Code Samples from Data Pruning

We show examples from our data pruning. The selected data samples adheres closely to the downstream coding tasks, from English problem description to code generation. We also provide an example of removed data from our pruning strategy.

Input

You are given a Python function that takes in a list of integers and a target integer. The function is intended to find two distinct elements in the list that sum up to the target integer. However, the function is incomplete and needs to be filled in with the appropriate code to achieve this functionality.
Complete the Python function ‘find_two_sum‘ to find and return the indices of two distinct elements in the input list that sum up to the target integer. If no such pair exists, return an empty list. The function signature is:

```
def find_two_sum(nums: List[int], target: int)
↪ -> List[int]:
```

For example, given the input ‘nums = [2, 7, 11, 15]’ and ‘target = 9’, the function should return ‘[0, 1]‘ since ‘nums[0] + nums[1] = 2 + 7 = 9’.

Output

```
from typing import List

def find_two_sum(nums: List[int], target: int)
↪ -> List[int]:
    num_indices = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_indices:
            return [num_indices[complement], i]
        num_indices[num] = i
    return []
```

Figure 6: Example of a selected data sample.

Input
You are given a list of integers representing the daily stock prices of a company. You need to write a function to calculate the maximum profit that can be obtained by buying and selling the stock at most once. If it is not possible to make any profit, return 0.
Write a function ‘maxProfit’ that takes in a list of integers ‘prices‘ as input and returns the maximum profit that can be obtained.
Function signature:

```
def maxProfit(prices: List[int]) -> int:
```

Example:

```
Input: prices = [7, 1, 5, 3, 6, 4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6 - 1 = 5.

Input: prices = [7, 6, 4, 3, 1]
Output: 0
Explanation: In this case, no transaction is done, i.e., max profit = 0.
```

Assumptions:
- The input list ‘prices’ will have at least one element.
- The stock prices are given in chronological order.

Output

```
from typing import List

def maxProfit(prices: List[int]) -> int:
    if not prices:
        return 0

    min_price = prices[0]
    max_profit = 0

    for price in prices:
        if price < min_price:
            min_price = price
        else:
            max_profit = max(max_profit, price - min_price)

    return max_profit
```

The ‘maxProfit’ function iterates through the list of prices, keeping track of the minimum price encountered so far and the maximum profit that can be obtained. It updates these values as it iterates through the list and returns the maximum profit at the end.

Figure 7: Another example of a selected data sample.

Input

````
Background:
You are a Product Manager, named Alice, your goal is Efficiently create a successful product, and the constraint is .
Question:

# Context
## Original Requirements
[BOSS: Write a podcast search engine which search for all index podcast. Each podcast belongs to a channel and each podcast contain title, description, published, audio link, transcript text]

## Search Information
### Search Results
### Search Summary

## mermaid quadrantChart code syntax example. DONT USE QUOTO IN CODE DUE TO INVALID SYNTAX. Replace the <Campain X> with REAL COMPETITOR NAME

quadrantChart
   title Reach and engagement of campaigns
   x-axis Low Reach --> High Reach
   y-axis Low Engagement --> High Engagement
...
````

Output

````
## Original Requirements
We are tasked with creating a podcast search engine that indexes all podcasts. Each podcast is associated with a channel and contains a title, description, publication date, audio link, and transcript text.

## Product Goals
```python
[
   "Create a user-friendly podcast search engine that indexes all podcasts",
   "Ensure each podcast contains necessary information: title, description, publication date, audio link, and transcript text",
   "Develop a system that efficiently updates and maintains the podcast index"
]
```

## User Stories
```python
[
   "As a user, I want to search for podcasts by title or description so that I can find the content I'm interested in",
   "As a user, I want to see all the necessary information about a podcast (title, description, publication date, audio link, transcript text) so that I can decide whether to listen to it",
   "As a user, I want the search engine to suggest podcasts based on my search history so that I can discover new content",
   "As a user, I want the podcast search engine to be updated regularly so that I can access the latest podcasts",
   "As a user, I want to filter search results by publication date, channel, or transcript text so that I can narrow down my search"
]
```

## Competitive Analysis
...
````

Figure 8: Example of a removed data sample (outlier).